



IAM PROJECT

CLOUD COMPUTING



Kubernetes Engine

Index

ABOUT ME	2
INTRODUCTION	3
OUR INFRASTRUCTURE	4
UNDERSTANDING WHAT IS KUBERNETES	7
OBJECTS IN KUBERNETES	8
WHY HAVE WE CHOSEN CLOUD TO BUILD OUR INFRASTRUCTURE?	11
CREATING OUR INSTANCES IN GCP PLATFORM FOR OUR PROJECT	13
INSTALLING NECESSARY PACKAGES + DOCKER + KUBERNETES	17
QUICK EXPLANATION ABOUT NETWORKS IN GCP	19
CREATING THE CLUSTER WITH KUBEADM	21
CREATING OUR INFRASTRUCTURE WITH HIGH AVAILABILITY	28
KUBERNETES ENGINE (AUTOSCALING GROUP)	30
CREATION OF OUR PERSISTENT FILE SYSTEM WITH GLUSTERFS	38
HEKETI INSTALLATION	41
CONFIGURATION OF KUBERNETES TO ACCESS HEKETI-CLI	46
DEPLOYING WORDPRESS	46
INSTALL CERT-MANAGER WITH HELM	48
CREATE INGRESS CONTROLLER WITH TRAEFFIK	49
CREATE THE HTTP/S LOADBALANCER	50
MONITORING OF SYSTEMS	51
MONITORING THE MAIN SERVICES WITH MONIT (EXTRA)	51
MONITORING ALL THE INFRASTRUCTURE WITH GRAFANA METRICS	54
DEPLOYING PROMETHEUS ON KUBERNETES CLUSTER	58
GRAFANA CONFIGURATION WITH DASHBOARDS	58
CONCLUSIONS	61
WEBGRAPHY	62

ABOUT ME

Hello! My name is Daniel Cascales and I am 21 years old. I am a systems administrator and currently I work in a company as a cloud systems administrator. Passionate about opensource technologies. In the following project you can see some of the most powerful / used technologies as a DevOps, Sysadmin.

Linkedin: <https://www.linkedin.com/in/dani-cascales-romero-b84240177/>

Github: <https://github.com/nanih98>

Email: a16dancasrom@iam.cat

Web (in progress, not available 4/06/2019): <https://wordpress-iamcloud.ddns.net/>

INTRODUCTION

Our project consists of a possible infrastructure based on Kubernetes that can be used by any company / institution using Open Source technologies, in this way we not only save money but we can help the community to develop new methods simpler and more efficient.

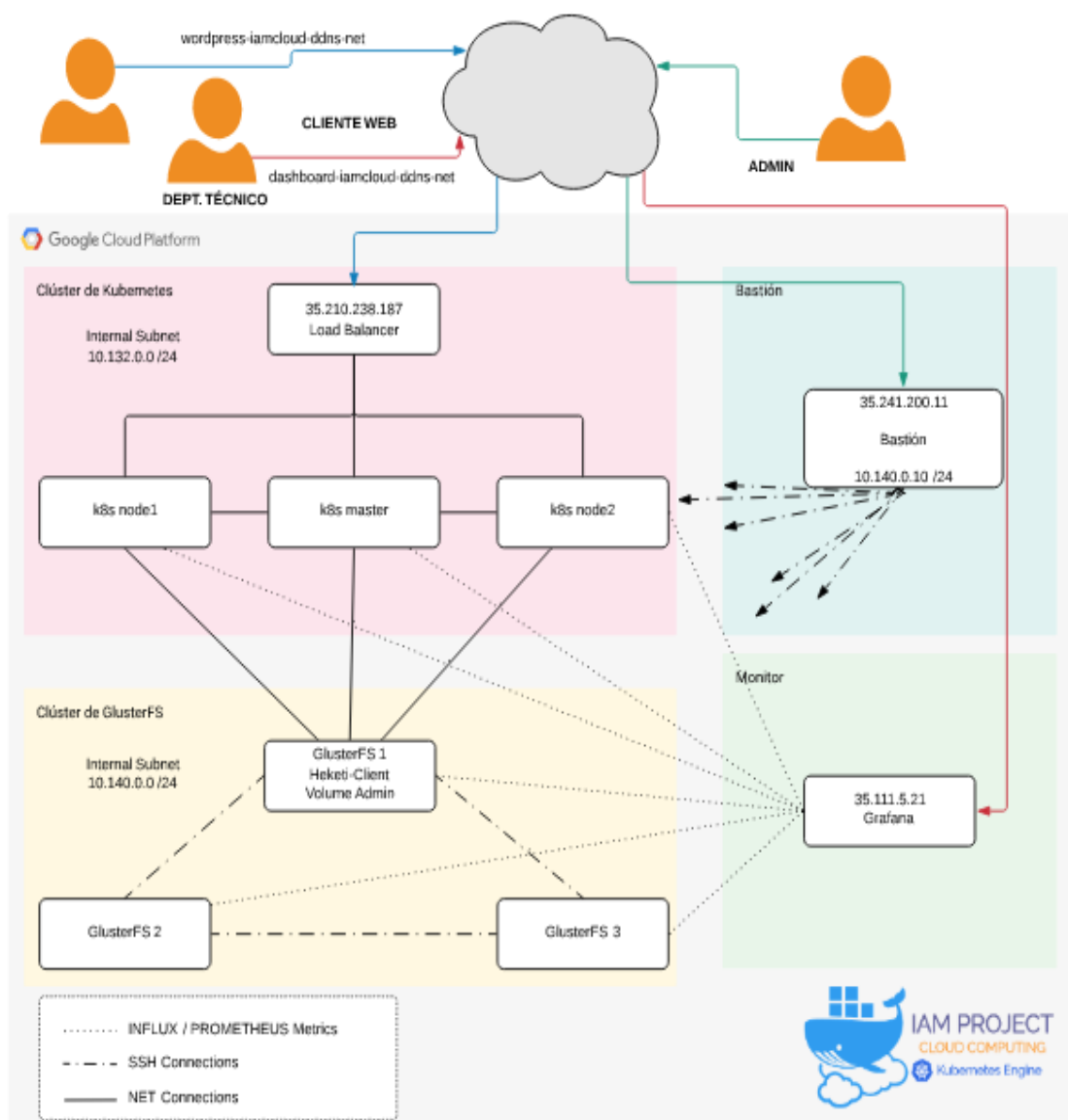
In this case we have decided to use the Google Cloud Platform, although Kubernetes is supported by any platform, it can even be implemented locally.

We use a Cloud platform mainly because the platforms that exist offer a very high availability in all the instances and services that we have to use, on the other hand we will forget to have facilities dedicated exclusively to maintain a high availability of our infrastructure, therefore not only saves us money but also a lot of time when it is time to lift it and put it into production.

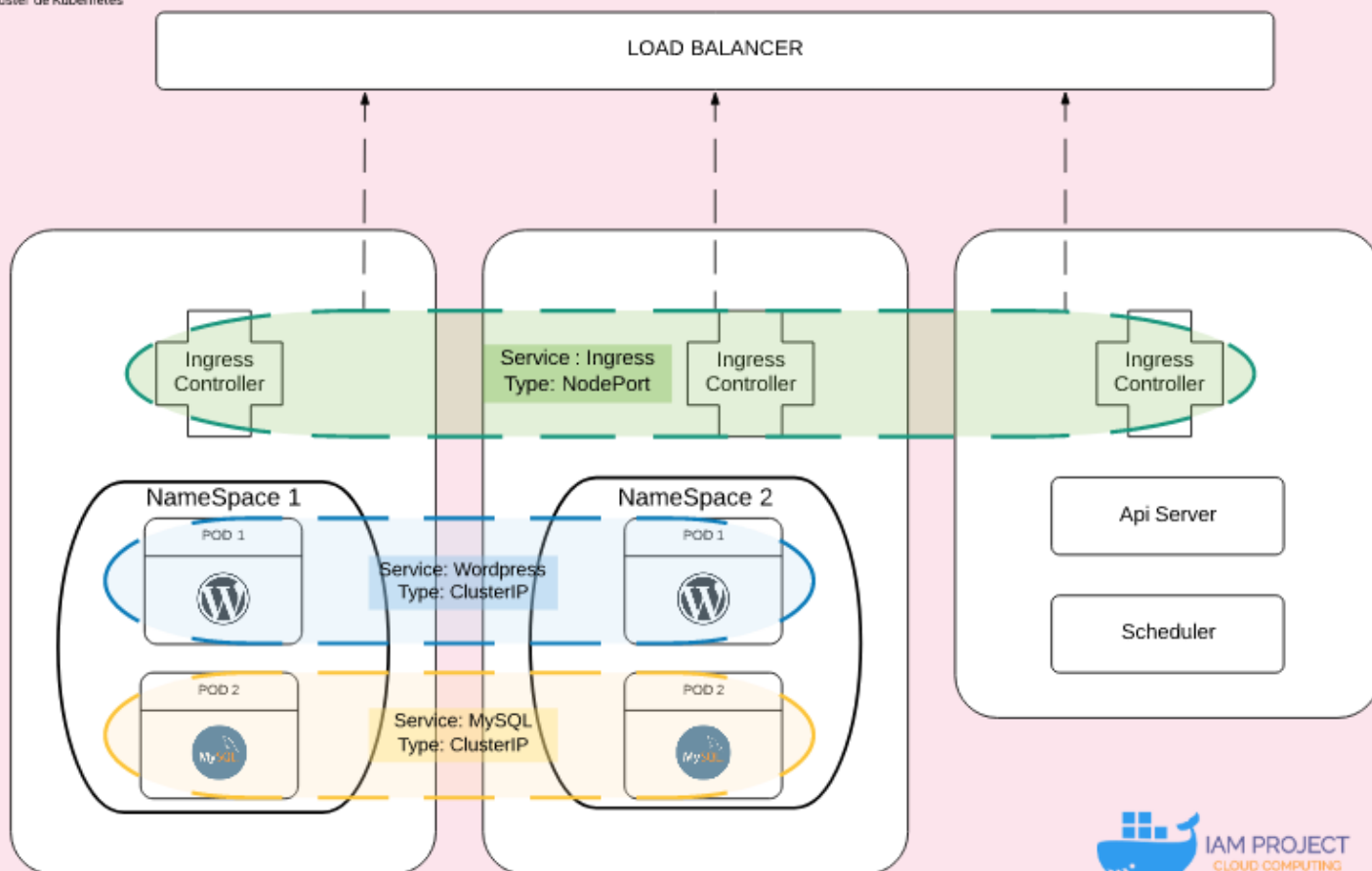
Due to Kubernetes we can automate the deployments, the escalation and the management of containerized applications, replace or replanize containers when a node dies. This gives us a very high availability and a very high percentage of automation of the entire infrastructure.

Once the infrastructure is deployed, it needs to be administered in the face off any type of failure. For this we have implemented a monitoring system based on state metrics with Prometheus and Telegraf, all centralized in the same platform using Grafana.

OUR INFRASTRUCTURE



Clúster de Kubernetes



UNDERSTANDING WHAT IS KUBERNETES

Kubernetes is an open-source container-orchestration system for automating application deployment, scaling, and management. It was originally designed by Google, and is now maintained by the Cloud Native Computing Foundation. It aims to provide a "platform for automating deployment, scaling, and operations of application containers across clusters of hosts". It works with a range of container tools, including Docker. Many cloud services offer a Kubernetes-based platform or infrastructure as a service (PaaS or IaaS) on which Kubernetes can be deployed as a platform-providing service. Many vendors also provide their own branded Kubernetes distributions.

Why did we choose Kubernetes?

- Scaling and autoscaling: depending on the CPU usage, it allows the vertical scaling of your applications automatically or manually (through a command or through the interface).
- Discovery of services and load balancing: it is not necessary to use an external mechanism for the discovery of services since Kubernetes assigns the containers their own IP addresses and a unique DNS name for a set of containers and can balance the load on them.
- Self-repair: in case of container failure you can restart it automatically. You can replace or re-plan containers when a node dies. And if there are containers that do not respond to health checks defined by the user, you can stop them.
- Automatic rollbacks and deployments: when an application needs to be updated or its settings changed, Kubernetes deploys the changes progressively while monitoring its health to ensure that it does not kill all instances at once, and in case of failure, it automatically rolls back
- Planning: it is responsible for deciding in which node each container will be executed according to the resources it requires and other restrictions. Mix critical workloads and best-effort to enhance the use and saving resources.
- Management of the configuration and secrets: sensitive information, such as passwords or ssh keys, is stored in Kubernetes hidden in secrets. Both the configuration of the application and the secrets are deployed and updated without having to reconstruct the image or expose confidential information.
- Storage orchestration: you can automatically mount the necessary storage system, be it local storage, storage in a public cloud provider (such as GCP or AWS), or even a network storage system such as NFS, SCSI, Gluster, Ceph, Cinder, or Flocker.
- Ejecución Batch: además de los servicios, Kubernetes puede gestionar cargas de trabajo batch y CI, reemplazando los contenedores que fallen.

OBJECTS IN KUBERNETES

Kubernetes Objects are persistent entities in the cluster. These objects are used to represent the state of the cluster.

The following are some of the Kubernetes Objects:

- **pods**
- **Namespaces**
- **ReplicationController (Manages Pods)**
- **DeploymentController (Manages Pods)**
- **StatefulSets**
- **DaemonSets**
- **Services**
- **ConfigMaps**
- **Volumes**

And then, let's explain a little the main objects that we have used:

Pod: a kubernetes pod is a group of containers that are deployed together on the same host. If you frequently deploy single containers, you can generally replace the word "pod" with "container" and accurately understand the concept.

Namespaces: it is designed so that you can organize your objects in a single namespace. For example, you can group in a single namespaces deployments, pods, services, ingress .. etc

ReplicationController: a ReplicationController ensures that a specified number of pod replicas are running at any one time. In other words, a ReplicationController makes sure that a pod or a homogeneous set of pods is always up and available.

DeploymentController: a DeploymentController provides declarative updates for Pods and ReplicaSets. You describe a desired state in a Deployment object, and the Deployment controller changes the actual state to the desired state at a controlled rate.

StatefulSets: statefulSet is the workload API object used to manage stateful applications. Manages the deployment and scaling of a set of Pods , and provides guarantees about the ordering and uniqueness of these Pods. Like a Deployment , a StatefulSet manages Pods that are based on an identical container spec. Unlike a Deployment, a StatefulSet maintains a sticky identity for each of their Pods. These pods are created from the same spec, but are not interchangeable: each has a persistent identifier that it maintains across any rescheduling.

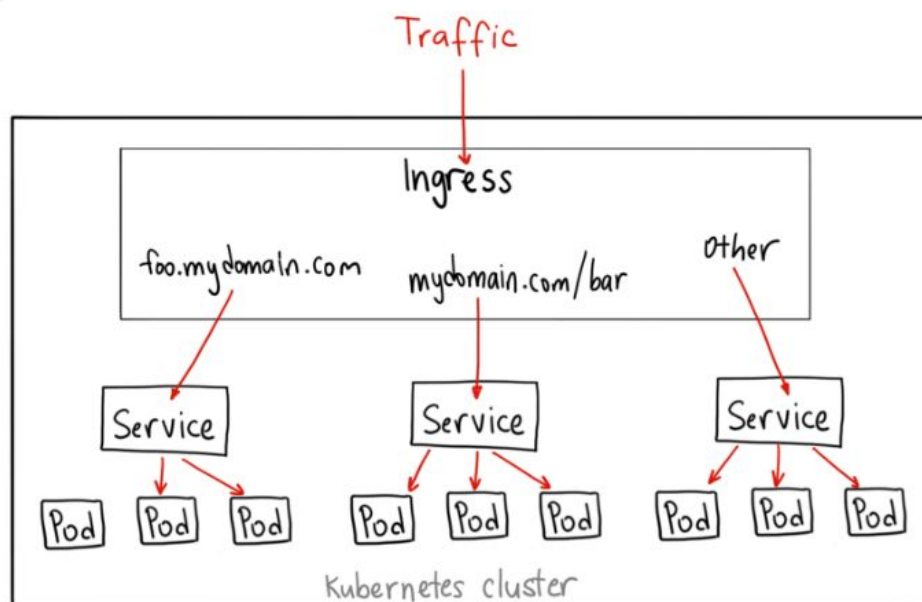
DaemonSets: a DaemonSet ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created.

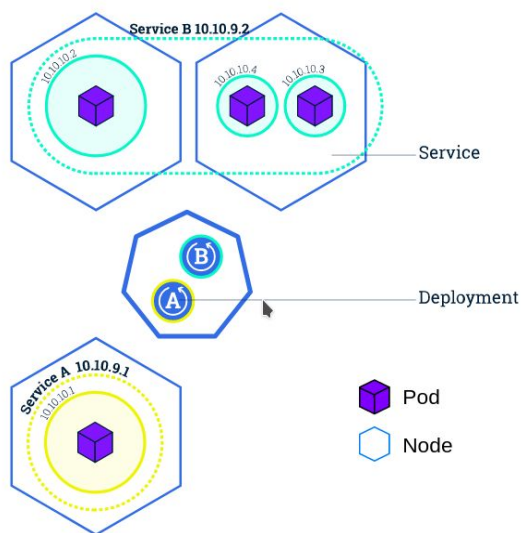
Services: a Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them - sometimes called a micro-service. The set of Pods targeted by a Service is (usually) determined by a Label Selector (see below for why you might want a Service without a selector).

Configmaps: configmaps links the configuration files, command line arguments, environment variables, port numbers and other configuration artifacts to the containers and system components of your pods in the runtime environment. ConfigMaps allows you to separate your configurations from your pods and components, which helps keep your workloads portable, makes your configurations easier to change and manage, and avoids coding configuration data according to pod specifications.

Volumes: on-disk files in a Container are ephemeral, which presents some problems for non-trivial applications when running in Containers. First, when a Container crashes, kubelet will restart it, but the files will be lost - the Container starts with a clean state. Second, when running Containers together in a Pod it is often necessary to share files between those Containers. The Kubernetes Volume abstraction solves both of these problems.

Ingress: an API object that manages external access to the services in a cluster, typically HTTP.





*In the previous image, we can see how two pods run in different nodes. These pods are exposed as a service. For the creation of these pods, a deployment has been created.

WHY HAVE WE CHOSEN CLOUD TO BUILD OUR INFRASTRUCTURE?

The reasons are as follows:

1. Flexibility

Cloud-based services are ideal for businesses with growing or fluctuating bandwidth demands. If your needs increase it's easy to scale up your cloud capacity, drawing on the service's remote servers. Likewise, if you need to scale down again, the flexibility is baked into the service. This level of agility can give businesses using cloud computing a real advantage over competitors – it's not surprising that CIOs and IT Directors rank 'operational agility' as a top driver for cloud adoption.

2. Disaster recovery

Businesses of all sizes should be investing in robust disaster recovery, but for smaller businesses that lack the required cash and expertise, this is often more an ideal than the reality. Cloud is now helping more organisations buck that trend. According to Aberdeen Group, small businesses are twice as likely as larger companies to have implemented cloud-based backup and recovery solutions that save time, avoid large up-front investment and roll up third-party expertise as part of the deal.

3. Automatic software updates

The beauty of cloud computing is that the servers are off-premise, out of sight and out of your hair. Suppliers take care of them for you and roll out regular software updates – including security updates – so you don't have to worry about wasting time maintaining the system yourself. Leaving you free to focus on the things that matter, like growing your business.

4. Capital-expenditure Free

Cloud computing cuts out the high cost of hardware. You simply pay as you go and enjoy a subscription-based model that's kind to your cash flow. Add to that the ease of setup and management and suddenly your scary, hairy IT project looks a lot friendlier. It's never been easier to take the first step to cloud adoption.

5. Increased collaboration

When your teams can access, edit and share documents anytime, from anywhere, they're able to do more together, and do it better. Cloud-based workflow and file sharing apps help them make updates in real time and gives them full visibility of their collaborations.

6. Work from anywhere

With cloud computing, if you've got an internet connection you can be at work. And with most serious cloud services offering mobile apps, you're not restricted by which device you've got to hand.

The result? Businesses can offer more flexible working perks to employees so they can enjoy the work-life balance that suits them – without productivity taking a hit. One study reported that 42% of workers would swap a portion of their pay for the ability to telecommute. On average they'd be willing to take a 6% pay cut.

7. Document control

The more employees and partners collaborate on documents, the greater the need for watertight document control. Before the cloud, workers had to send files back and forth as email attachments to be worked on by one user at a time. Sooner or later – usually sooner – you end up with a mess of conflicting file content, formats and titles.

And as even the smallest companies become more global, the scope for complication rises. According to one study, "73% of knowledge workers collaborate with people in different time zones and regions at least monthly".

When you make the move to cloud computing, all files are stored centrally and everyone sees one version of the truth. Greater visibility means improved collaboration, which ultimately means better work and a healthier bottom line. If you're still relying on the old way, it could be time to try something a little more streamlined.

8. Security

Lost laptops are a billion dollar business problem. And potentially greater than the loss of an expensive piece of kit is the loss of the sensitive data inside it. Cloud computing gives you greater security when this happens. Because your data is stored in the cloud, you can access it no matter what happens to your machine. And you can even remotely wipe data from lost laptops so it doesn't get into the wrong hands.

9. Competitiveness

Wish there was a simple step you could take to become more competitive? Moving to the cloud gives access to enterprise-class technology, for everyone. It also allows smaller businesses to act faster than big, established competitors. Pay-as-you-go service and cloud business applications mean small outfits can run with the big boys, and disrupt the market, while remaining lean and nimble. David now packs a Goliath-sized punch.

10. Environmentally friendly

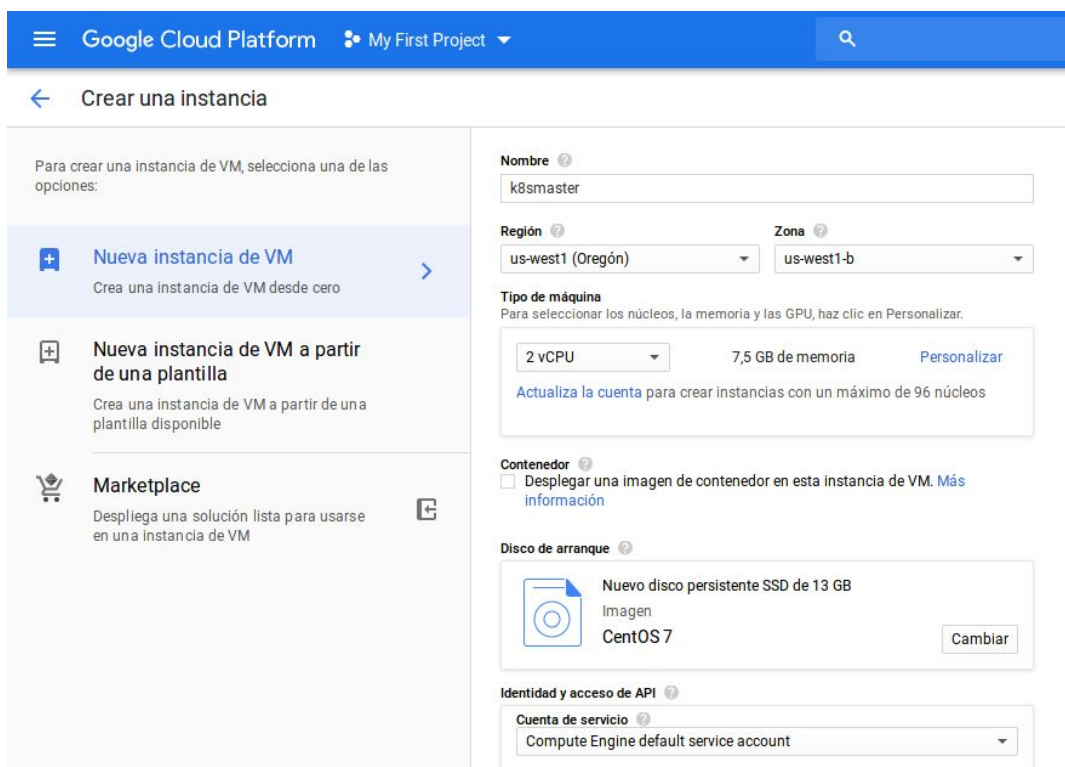
While the above points spell out the benefits of cloud computing for your business, moving to the cloud isn't an entirely selfish act. The environment gets a little love too. When your cloud needs fluctuate, your server capacity scales up and down to fit. So you only use the energy you need and you don't leave oversized carbon footprints. This is something close to our hearts at Salesforce, where we try our best to create sustainable solutions with minimal environmental impact.

CREATING OUR INSTANCES IN GCP PLATFORM FOR OUR PROJECT

Let's create 3 manually instances in our google cloud account, 1 for master and the other for workers (nodes).

NOTE: the language of the images is in Spanish, but it is also understandable in any other language.

Log in to Google Cloud and go to Compute Engine -> Instances -> Create an instance



Google Cloud Platform My First Project

Crear una instancia

Para crear una instancia de VM, selecciona una de las opciones:

- Nueva instancia de VM**
Crea una instancia de VM desde cero
- Nueva instancia de VM a partir de una plantilla**
Crea una instancia de VM a partir de una plantilla disponible
- Marketplace**
Despliega una solución lista para usarse en una instancia de VM

Nombre
k8smaster

Región
us-west1 (Oregón)

Zona
us-west1-b

Tipo de máquina
Para seleccionar los núcleos, la memoria y las GPU, haz clic en Personalizar.
2 vCPU 7,5 GB de memoria Personalizar
Actualiza la cuenta para crear instancias con un máximo de 96 núcleos

Contenedor
☐ Desplegar una imagen de contenedor en esta instancia de VM. Más información

Disco de arranque
Nuevo disco persistente SSD de 13 GB
Imagen CentOS 7 Cambiar

Identidad y acceso de API
Cuenta de servicio
Compute Engine default service account

NOTE: we need to create machines with at least 2 cores, otherwise Kubernetes will not work. Haproxy can work only with 1 core.

I will use the centos 7 operating system, but any Linux distro can be used.

WARNING: obviously we can install machines with more power, but for a project that will not be put into production is more than enough. We must remember that we only have \$ 300 of free Google Cloud credit. Having so many machines you have to be careful so that they do not overcharge us.

Specifications:

Hostname	RAM	CPU	DISK	Image
K8sMaster	7.5GB	2 cores	13GB SSD	Centos 7
K8sNode1	7.5GB	2 cores	13GB SSD	Centos 7
K8sNode2	7.5GB	2 cores	13GB SDD	Centos 7

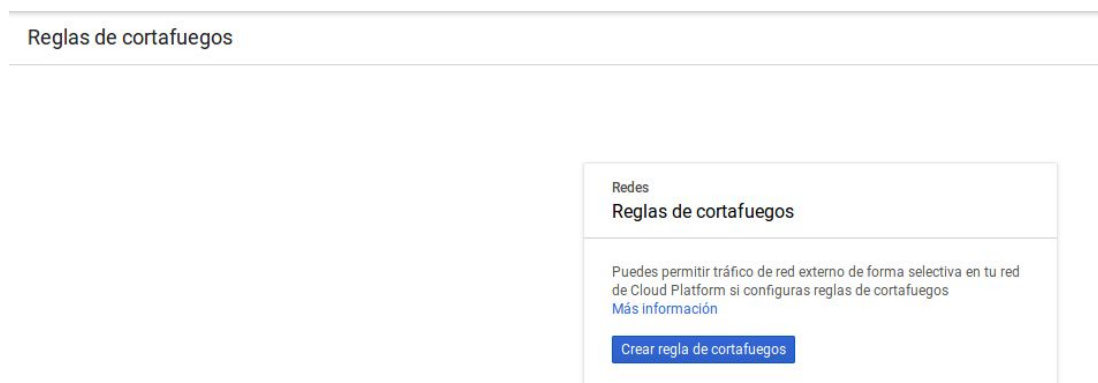
Finally we must have something like that:



Nombre	Zona	Recomendación	Usada por	IP interna	IP externa	Conectar
haproxy	us-west1-b			10.138.0.34 (nic0)	34.83.173.174	SSH
k8smaster	us-west1-b			10.138.0.31 (nic0)	35.197.0.234	SSH
k8snode2	us-west1-b			10.138.0.33 (nic0)	35.227.183.240	SSH
ks8node1	us-west1-b			10.138.0.32 (nic0)	35.247.34.46	SSH

NOTE: we have reduced space in the nodes since we will soon install another server that will make NFS for persistent volumes. **The server that you see called haproxy, do not pay attention to it, it's just a test server.**

Now, we need to create the ssh rules for allow connections to our machines. Go to network of VPC→firewall rules. Click on the icon “Create new firewall rule”.



Reglas de cortafuegos

Redes
Reglas de cortafuegos

Puedes permitir tráfico de red externo de forma selectiva en tu red de Cloud Platform si configuras reglas de cortafuegos
[Más información](#)

[Crear regla de cortafuegos](#)

Only we need to change the following parameters:

Crear una regla de cortafuegos

Las reglas de cortafuegos controlan el tráfico entrante o saliente de una instancia. De forma predeterminada, se bloquea el tráfico entrante que sea ajeno a tu red. [Más información](#)

Nombre

ssh

Descripción (Opcional)

Firewall ssh rules for our instances connections.

Add the ip from which we will allow the connection. Also we must apply a label to the firewall rule to apply later to the instances.

Etiquetas de destino

ssh

Filtro de origen

Intervalos de IPs

Intervalos de IPs de origen

84.7....etc |

El intervalo o la dirección IP no son válidos. Usa la notación CIDR y escribe la dirección IP más baja de la subred.

Finally click on the icon “Create” to apply the changes.

Now, apply that firewall labels to the instances. Go to the instance panel, click on the instance, for example ‘k8smaster’ and click “Edit” button. Go to the section network labels and put the label, in that case, the ssh label. **Do the same with the other instances.**

Cortafuegos

☐ Permitir el tráfico HTTP

☐ Permitir el tráfico HTTPS

Etiquetas de red

ssh |

Now, we can connect to our instances using the **PUBLIC IP** that we specified in the firewall rule 'SSH'. First of all, it's necessary the package GOOGLE CLOUD SDK.

Follow the instructions of the following link to install it:

<https://cloud.google.com/sdk/install>

Now, go another time to the instance panel and click the following button to get the gcloud command:

<input type="checkbox"/>	Nombre ^	Zona	Recomendación	Usada por	IP interna	IP externa	Conectar
<input type="checkbox"/>	✓ haproxy	us-west1-b			10.138.0.34 (nic0)	34.83.173.174	SSH ⌵
<input type="checkbox"/>	✓ k8smaster	us-west1-b					⋮
<input type="checkbox"/>	✓ k8snode2	us-west1-b					⋮
<input type="checkbox"/>	✓ ks8node1	us-west1-b					⋮

Abrir en la ventana del navegador

Abrir en una ventana de navegador en un puerto personalizado

Abrir en una ventana de navegador con la clave SSH privada suministrada

Ver comando gcloud

Utilizar otro cliente de SSH

Click on '**See gcloud command**' and copy it. Paste in your terminal.



Necessary steps:

\$ gcloud auth login #Necessary for authenticate gmail account with gcloud.

· Create the passphrase for the connection. Is something like when you use ssh connection like: ssh -i key.pem user@ip ... but google has it own bash.

```
anonymous@anonymous-ackstorm 12:17:47
> gcloud compute --project "avian-cogency-237220" ssh --zone "us-west1-b" "haproxy"
Warning: Permanently added 'compute.4898881670426981529' (ECDSA) to the list of known hosts.
Enter passphrase for key '/home/anonymous/.ssh/google_compute_engine':
[anonymous@haproxy ~]$
```


INSTALLING NECESSARY PACKAGES + DOCKER + KUBERNETES

Connect to the master,node1 and node2 instances and run the following script:

```
$ sudo su
$ touch preinstall.sh
$ nano preinstall.sh # Copy the script inside this file.
```

```
[anonymous@k8smaster ~]$ sudo su
[root@k8smaster anonymous]# vim preinstall.sh
[root@k8smaster anonymous]# bash preinstall.sh
```

* It will take a couple of minutes

```
#!/bin/bash
#Preinstallation, docker and kubernetes services installation for centos instance.
yum -y update && yum -y update && yum -y install make gcc kernel-devel kernel-headers net-tools vim
#Disable selinux
setenforce 0
sed -i --follow-symlinks 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/selinux
modprobe br_netfilter
echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables
#Disable swap, it's required for kubernetes
swapoff -a
#Install necessary packages for docker
yum install -y yum-utils device-mapper-persistent-data lvm2
#Add repo
yum-config-manager \
--add-repo \
https://download.docker.com/linux/centos/docker-ce.repo
#Install docker
yum install -y docker-ce docker-ce-cli containerd.io docker-compose
#Enable the service and start it.
systemctl start docker && systemctl enable docker
#Allow docker command only for root
usermod -aG docker $(whoami)
#Add kubernetes repos
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
```

```
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
      https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF
#Install kubernetes
yum install -y kubelet kubeadm kubectl
#Enable kubernetes service and start it
systemctl start kubelet && systemctl enable kubelet
```

Once installed, we check that the services are working:

```
[root@k8smaster anonymous]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since vie 2019-05-10 10:47:07 UTC; 54s ago
     Docs: https://docs.docker.com
    Main PID: 27606 (dockerd)
    CGroup: /system.slice/docker.service
            └─27606 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

may 10 10:47:06 k8smaster dockerd[27606]: time="2019-05-10T10:47:06.385362073Z" level=info msg="pic
```

```
[root@k8smaster anonymous]# systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
   Loaded: loaded (/usr/lib/systemd/system/kubelet.service; disabled; vendor preset: disabled)
   Drop-In: /usr/lib/systemd/system/kubelet.service.d
            └─10-kubeadm.conf
   Active: activating (auto-restart) (Result: exit-code) since vie 2019-05-10 10:48:20 UTC; 2s ago
     Docs: https://kubernetes.io/docs/
   Process: 27856 ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBELET_KUBI
```

NOTE: the Kubernetes service will start working when we create the cluster. At the moment there is no need to worry.

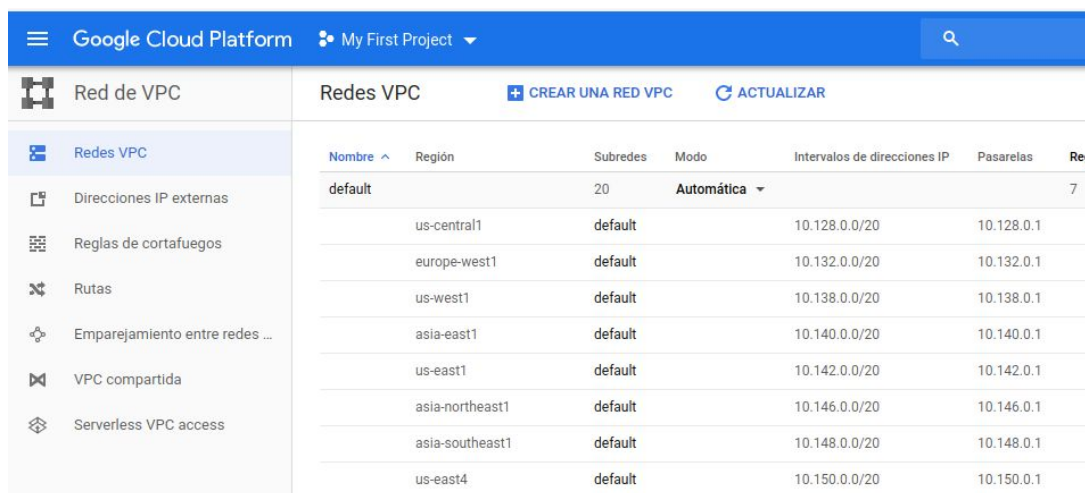
QUICK EXPLANATION ABOUT NETWORKS IN GCP

Before starting to work with the instances it would be convenient to understand how the networks work in GCP. The part of the public ip I think that everyone understands it. GCP assigns you a public ip for each instance that you create to be able to connect to it, SSH, http / http through haproxy..etc

On the other hand, GCP gives you the opportunity to create your own virtual network in the cloud, which will use your machines to be able to see each other. It's a subject that I'm not going to touch now. I will use the 'default' network assigned to you depending on where your instance is located.

Example:

My instances are in Oregon, and Oregon's default private network is 10.138.0.0/16. Therefore, for this project they must all be in the same private network, otherwise, they will not be seen and will not work.



Nombre	Región	Subredes	Modo	Intervalos de direcciones IP	Pasarelas	Req
default		20	Automática			7
	us-central1	default		10.128.0.0/20	10.128.0.1	
	europa-west1	default		10.132.0.0/20	10.132.0.1	
	us-west1	default		10.138.0.0/20	10.138.0.1	
	asia-east1	default		10.140.0.0/20	10.140.0.1	
	us-east1	default		10.142.0.0/20	10.142.0.1	
	asia-northeast1	default		10.146.0.0/20	10.146.0.1	
	asia-southeast1	default		10.148.0.0/20	10.148.0.1	
	us-east4	default		10.150.0.0/20	10.150.0.1	

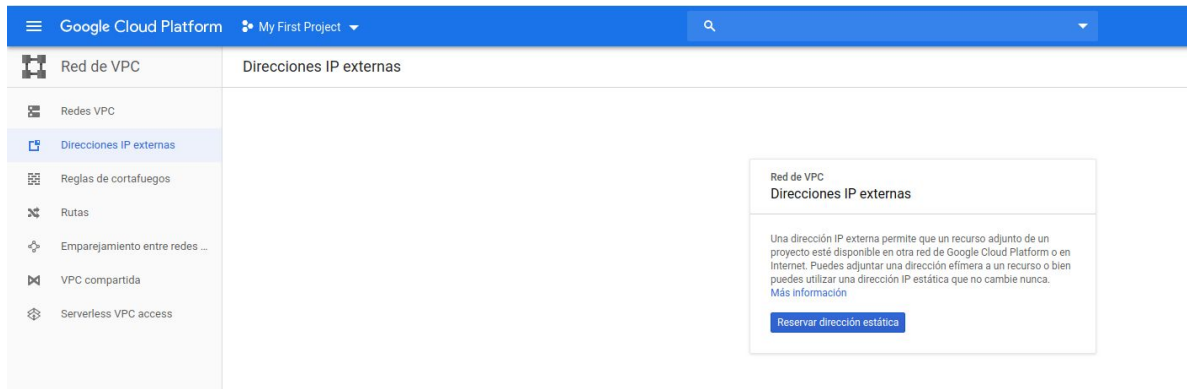
*As you can see in the previous image, we can see the default networks that exist in different places.

NOTE: the place where your instance is created is chosen at the moment you create it, from there, your private network will be one or the other as I have explained.

<input type="checkbox"/>	Nombre	Zona	Recomendación	Usada por	IP interna	IP externa	Conectar
<input type="checkbox"/>	haproxy	us-west1-b			10.138.0.41 (nic0)	Ninguna	SSH
<input type="checkbox"/>	k8smaster	us-west1-b			10.138.0.38 (nic0)	Ninguna	SSH
<input type="checkbox"/>	k8snode1	us-west1-b			10.138.0.39 (nic0)	Ninguna	SSH
<input type="checkbox"/>	k8snode2	us-west1-b			10.138.0.40 (nic0)	Ninguna	SSH
<input type="checkbox"/>	nfs	us-west1-b			10.138.0.43 (nic0)	Ninguna	SSH

*Example of the network of my instances.

IMPORTANT: when you turn off the instances, the public ips will change unless you reserve them. The private ip do not have to change, they are static. You can reserve public ip in the section of VPC network.



CREATING THE CLUSTER WITH KUBEADM

Intro:

There are many ways to create a cluster of kubernetes, the most 'healthy' way would be with kubernetes the hard way (I think), but as we are not experts at the moment, we will do it with kubeadm which is a very simple and understandable way.

When you create kubernetes the hard way, you are creating kubernetes as a service of the system itself, so it is better from my perspective. With kubeadm, kubernetes is created inside containers with docker, such that it is a bit cumbersome. In any case, it is functional and safe and will work perfectly. In fact, it is speculated that 'kubeadm' is the future when it comes to creating kubernetes clusters.

Follow the steps only in '**MASTER NODE**':

We will use the command '**kubeadm init**' and some necessary args:

1. Create the subnet for pods, with the parameter `--pod-network-cidr`. The .yaml with flannel **need to have the same network for running ok**.
2. The master ip (internal), by default is the ip of eth0, eno1 etc. If we need to change it add the arg: '`--apiserver-advertise-address= "IP" '`

What is the internal ip?

<input type="checkbox"/>	<input checked="" type="checkbox"/>	k8smaster	us-west1-b	10.138.0.31 (nic0)
--------------------------	-------------------------------------	-----------	------------	--------------------

What is the flannel.yaml file?

The flannel.yaml file is the "driver" that pods (containers) need to connect with each other. There are a lot of types of networks for kubernetes cluster.

More info: <https://kubernetes.io/docs/concepts/cluster-administration/networking/>

Then run the command:

```
$ kubeadm init --pod-network-cidr=10.244.0.0/16 --apiserver-advertise-address=10.138.0.31
```

```
[root@k8smaster anonymous]# kubeadm init --pod-network-cidr=10.244.0.0/16 --apiserver-advertise-address=10.138.0.31
[init] Using Kubernetes version: v1.14.1
[preflight] Running pre-flight checks
[WARNING Firewall]: firewalld is active, please ensure ports [6443 10250] are open or your cluster may not function
[WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recommended driver is "systemd"
[WARNING Service-Kubelet]: kubelet service is not enabled, please run 'systemctl enable kubelet.service'
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Activating the kubelet service
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [k8smaster kubernetes kubernetes.default kubernetes.default.svc kube
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [k8smaster localhost] and IPs [10.138.0.31 127.0.0.1 ::1]
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [k8smaster localhost] and IPs [10.138.0.31 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "sa" key and public key
```

When the process is over, we must be attentive to what it shows us:

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 10.138.0.31:6443 --token xq5r4u.jys4j4amn4omk5cm \
--discovery-token-ca-cert-hash sha256:e99fd199cd694b653688c60ea2d55f52f3babcad4567239ea012d31f7cd93fa4
[root@k8smaster anonymous]#
```

Save the kubeadm join command, because here we have the token to join the workers to the master. Copy it in a document to save it.

Example:

```
$ kubeadm join 10.138.0.31:6443 --token xq5r4u.jys4j4amn4omk5cm \
--discovery-token-ca-cert-hash
sha256:e99fd199cd694b653688c60ea2d55f52f3babcad4567239ea012d31f7cd93fa4
```

As a normal user, run:

*We need sudo privileges of the normal user.

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

```
exit
[anonymous@k8smaster ~]$ mkdir -p $HOME/.kube
[anonymous@k8smaster ~]$
[anonymous@k8smaster ~]$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
[anonymous@k8smaster ~]$
[anonymous@k8smaster ~]$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
[anonymous@k8smaster ~]$
```

Now we will add the flannel.yaml for the network of the pods, as we commented previously.

Run as normal user, too:

\$ kubectl apply -f

<https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml>

As we commented previously, the flannel.yaml network must be the same as the one we specified in the kubeadm init --pod-network-cidr command , so

How do we prove it?

Open the link

<https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml>

in the browser and search (CONTROL+F) 'network':

```
    },
    {
      "type": "portmap",
      "capabilities": {
        "portMappings": true
      }
    }
  ]
}
net-conf.json: |
{
  "Network": "10.244.0.0/16",
  "Backend": {
    "Type": "vxlan"
  }
}
---
```

As you can see, the network is the same. **10.244.0.0/16**

\$ kubeadm init --pod-network-cidr=10.244.0.0/16 ..etc

Creating new firewall rules to enable connection between nodes (kubernetes default ports):

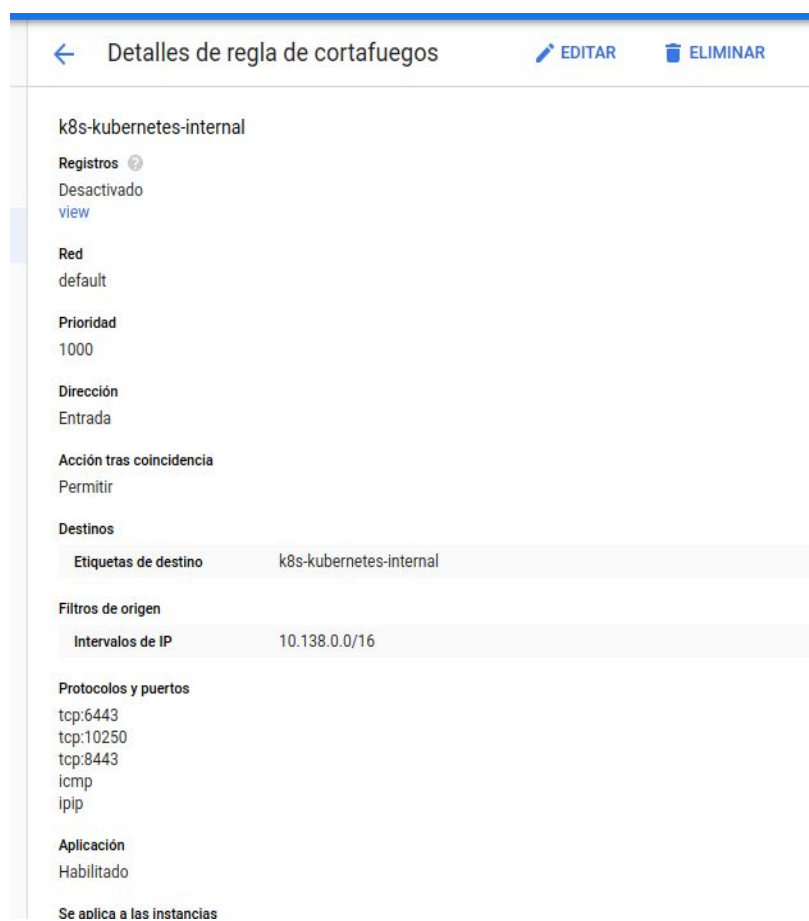
→ tcp:6443,10250,8443

→ icmp

→ ipip

*We must follow the steps as we did when creating ssh firewall rule.

See the follow picture, we need to have something like that:



NOTE: don't remember to apply that firewall rule label to the instances (master,node1,node2). Also this firewall rules only must be applied so that the nodes can communicate through the private ip (in my case is 10.138.0.0/16).

How do we join the workers to the master?

Now that we have the ports of kubernetes enabled, copy the kubeadm join command that we previously saved and run it in the node1 and node2 workers:

Example:

```
[root@k8snode2 anonymous]# kubeadm join 10.138.0.31:6443 --token xq5r4u.jys4j4amn4omk5cm --discovery-token-ca-cert-hash sha256:e99fd19
[preflight] Running pre-flight checks
[WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recommended driver is "systemd". Please follow the documentation to update to systemd.
[WARNING Service-Kubelet]: kubelet service is not enabled, please run 'systemctl enable kubelet.service'
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.14" ConfigMap in the kube-system namespace
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Activating the kubelet service
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
[root@k8snode2 anonymous]#
```

Check if the nodes are 'ready':

```
[anonymous@k8smaster ~]$ kubectl get nodes
NAME           STATUS    ROLES    AGE   VERSION
k8smaster      Ready    master   51m   v1.14.1
k8snode2       Ready    <none>   10m   v1.14.1
k8snode1       Ready    <none>   7m24s v1.14.1
[anonymous@k8smaster ~]$
```

Now copy the config of kubernetes to the workers:

`[user@k8smaster]:$ cat /home/your_user/.kube/config`

```
[anonymous@k8smaster ~]$ cat .kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVRVJS
Vd056RXhNRfExTmxvd0ZURVRNqkVHQTFVRQpBeE1LYTNWaVpYSnVa
YKaxRzcUlaMjgzQ0JZbVd3THVWQzVGa1F4dVR5MytCVklic2I0bmk
m1DSk9UaXdHaitQY2Y4T1Z1Y1hydTAxVkJj0FB0TnluaHZkVU5obE
Z0trTUE4R0ExVWRFd0VCCi93UUZNQU1CQWY4d0RRWUpLb1pJaHZjT
qRTd6WFdSRklFeEVjTFkrQ2cwQVJGRnptSTR0VGNVZi9rZWdwewVQ
Z2VG1oYw1ma31rT09tMW5mZnZCQXdhVzRwM3ZpWop3dW9lUkx1OTI
```

Why?

This allows to use the kubectl command in the workers:

*Do the same in each workers nodes:

```
[anonymous@k8snode2 ~]$ mkdir -p $HOME/.kube
[anonymous@k8snode2 ~]$ cd .kube/
[anonymous@k8snode2 .kube]$ touch config
[anonymous@k8snode2 .kube]$ ls
config
[anonymous@k8snode2 .kube]$ vim config
[anonymous@k8snode2 .kube]$ cd
[anonymous@k8snode2 ~]$ kubectl get nodes
NAME          STATUS    ROLES    AGE   VERSION
k8smaster     Ready     master   54m   v1.14.1
k8snode2      Ready     <none>   13m   v1.14.1
ks8node1      Ready     <none>   10m   v1.14.1
[anonymous@k8snode2 ~]$
```

```
[anonymous@ks8node1 ~]$ kubectl get nodes
NAME          STATUS    ROLES    AGE   VERSION
k8smaster     Ready     master   59m   v1.14.1
k8snode2      Ready     <none>   18m   v1.14.1
ks8node1      Ready     <none>   14m   v1.14.1
[anonymous@ks8node1 ~]$
```

Changing the label of worker nodes:

Go to the master terminal and run the following command:

\$ kubectl label node "node_name" node-role.kubernetes.io/"new_name"=

In our case, the command is:

\$ kubectl label node k8snode1 node-role.kubernetes.io/worker1=

\$ kubectl label node k8snode2 node-role.kubernetes.io/worker2=

Let's see if the changes have been applied:

```
[anonymous@k8smaster ~]$ kubectl get nodes
NAME          STATUS    ROLES    AGE   VERSION
k8smaster     Ready     master   12m   v1.14.1
k8snode1      Ready     worker1  11m   v1.14.1
k8snode2      Ready     worker2  11m   v1.14.1
[anonymous@k8smaster ~]$
```

CREATING OUR INFRASTRUCTURE WITH HIGH AVAILABILITY

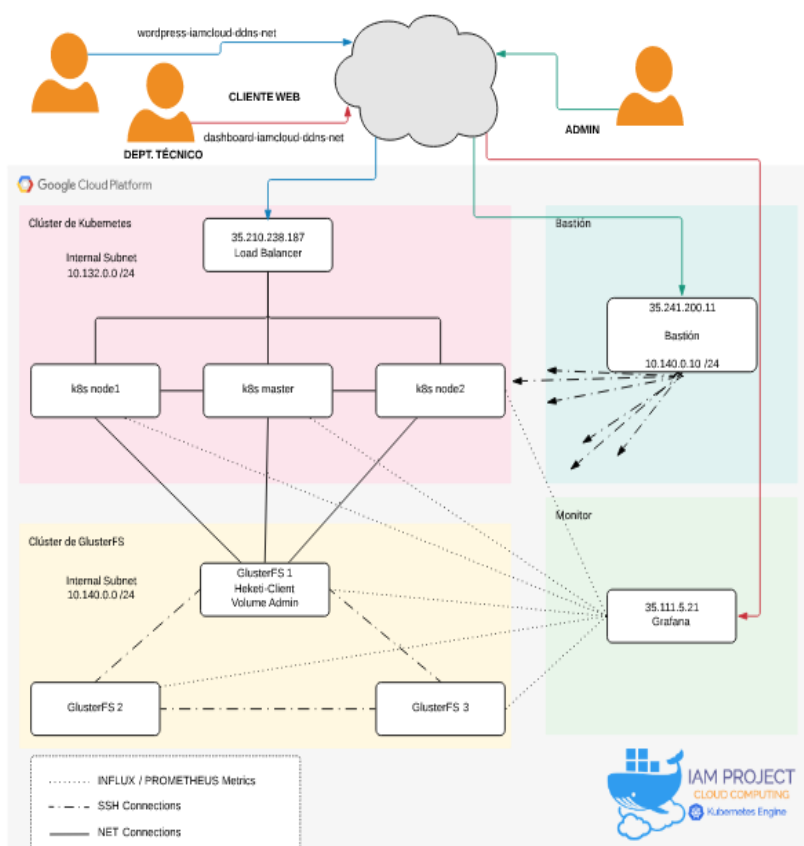
So far we have created the infrastructure 'by hand'. From this section, we will create the entire infrastructure by incorporating load balancers and autoscalers that google incorporates as a service. We need high availability because we must think that everything we are creating could be a real case of a client.

Why use these services of lb and autoscaling of google?

This type of high availability services can be created 'by hand', but you can not be sure that it is really sophisticated so that it works 100% when putting them into production. This is the reason why we choose google, since they are responsible for everything to work and you should not worry.

We will also incorporate a bastion to enable the ssh connection only from that machine. It is a safety measure to avoid connections from outside to our machines.

NOTE: therefore, the structure should be as follows, both bastion and the cluster of kubernetes as the cluster of glusterfs and the monitoring server should be **divided into 4 different subnets**.



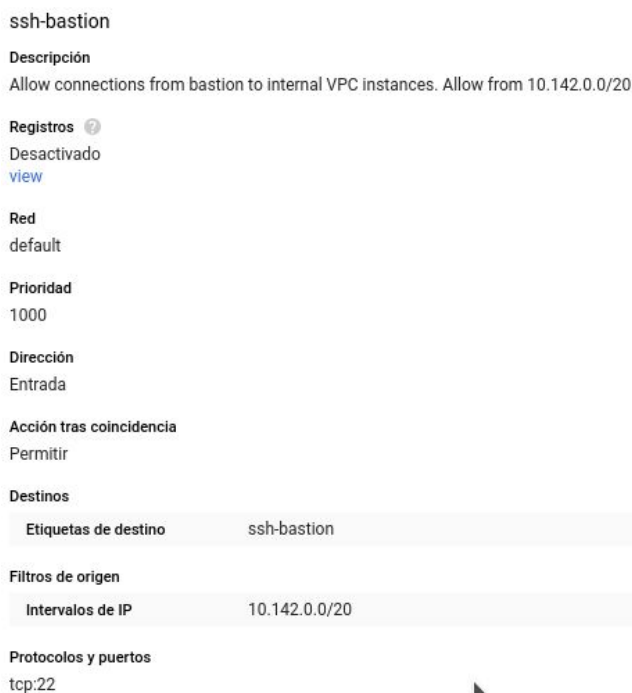
Specifications of the bastion machine:

Centos 7	10 GB -HDD	1 CORE	1 GB RAM
-----------------	-------------------	---------------	-----------------

Once we have our bastion machine created, we must create public-private ssh keys, save the public key, create the config file and create a firewall rule **to be applied later in the clusters and in the monitoring server**.

Create the firewall rule:

In the firewall rule 'ssh-bastion', we will simply specify that we will allow the ssh connection (tcp 22) from the bastion machine.



Subnet specifications:

Instances	Kubernetes-engine	Glusterfs cluster	Bastion	Monitor
Subnet	10.138.0.0/20	10.150.0.0/20	10.142.0.0/20	10.132.0.0/20

Config file for ssh:

`*/home/user/.ssh/config`

In the following link is the example of configuration file of the bastion server:

https://github.com/nanih98/iamcloud/blob/master/SSH/ssh_config_example.txt

IMPORTANT: to this machine we must enable the ssh connection from the outside, since it is our access point to the others.

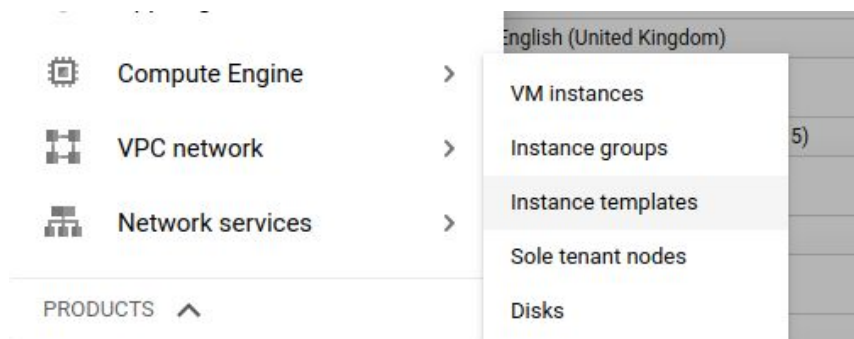
The firewall rule label is 'ssh' where we specify external public ip from where we can access. We have examples in the other sections.

KUBERNETES ENGINE (AUTOSCALING GROUP)

Steps:

1. Create our templates for the kubernetes instances.

Go to the section 'Compute Engine' > 'Instance Templates' :



Then click on 'Create instance template' :

We are going to create the templates for our kubernetes nodes. Basically they will be the same resources that in the previous section we configured (same cpu,ram...etc).

← Create an instance template

Describe a VM instance once and then use that template to create groups of identical instances. [Learn more.](#)


Name ⓘ
kubernetes-templates

Machine type ⓘ
Customise to select cores, memory and GPUs.

2 vCPUs 7.5 GB memory [Customise](#)
[Upgrade your account](#) to create instances with up to 96 cores

Container ⓘ
☐ Deploy a container image to this VM instance. [Learn more](#)

Boot disk ⓘ

 New 13 GB SSD persistent disk
Image
CentOS 7 [Change](#)

Identity and API access ⓘ

Service account ⓘ
Compute Engine default service account

Automation

Startup script (Optional)

You can choose to specify a startup script that will run when your instance boots up or restarts. Startup scripts can be used to install software and updates, and to ensure that services are running within the virtual machine. [Learn more.](#)

```
#!/bin/bash
#Preinstallation, docker and kubernetes services installation for centos
```

*In the previous image, we will add the script that we created in the section '**INSTALLING NECESSARY PACKAGES + DOCKER + KUBERNETES**' to supply the necessary services of the kubernetes cluster. Also the code is available on my github <https://github.com/nanih98/iamcloud/tree/master/Scripts> → **gcp_script_kubernetes_template.sh**

We will enable the ssh-bastion+kubernetes-internal firewall tag to enable the connections. We will assign the rest of the rules for the cluster later if necessary.

Administración Seguridad Discos Redes Único propietario

Red [?]
default

Subred [?]
Subred automática

Mostrar intervalos de IP de alias

Etiquetas de red [?] (Opcional)
ssh-bastion x kubernetes-internal x
kubernetes-engine-services-export x

IP externa [?]
Efímera

Nivel de servicio de red [?]
☒ Premium (nivel de proyecto actual, [cambiar](#)) [?]
☐ Estándar [?]

Reenvío de IP [?]
Desactivado

Paste the ssh public key to allow connections from 'bastion':

Administración
Seguridad
Discos
Redes
Único propietario

VM blindada

Para usar los roles de VM blindada, debes seleccionar una imagen blindada.

Para disfrutar de la configuración más segura, activa todos los ajustes.

☐ Activa el arranque seguro
☐ Activa el vTPM
☐ Activa la supervisión de integridad

Claves SSH

Estas claves permiten acceder únicamente a esta instancia, a diferencia de las [claves SSH del proyecto](#). [Más información](#)

☐ Bloquear claves SSH del proyecto

Si marcas esta opción, las claves SSH del proyecto no pueden acceder a esta instancia. [Más información](#)

anonymous

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCAQC7nNkH0aNJ/K6pB7Ub844iIwmTTB0aNsh/H0PMnULX7Ui3JQqqy314EhWy0RmsQCZEhvt8oaEtsxt9Zc1c2TWXhdn86wUu/82yvoRM0hXhCOiWKqpodhb1x091Q+cCj9bwGJUfo00uksUKx4NkOjOoOfgDWyj35nkraYN1o/PirrYjxMBQY5no+GDv1MDEFeCuapveXjI/19u3Izw1rR2mrQPbHk1p1/BiEioF1vciYwQ0/TKQaaJFXbmTpVN1A

×


+ Añadir elemento


NOTE: the kubernetes-internal firewall rule is the rule that we created in the previous section where we created the kubernetes cluster. In it, we specified the default ports of kubernetes to enable communication between nodes. See the section **‘CREATING THE CLUSTER WITH KUBEADM’** if you have any problem.

2. Create our google autoscaler with the instances of kubernetes.

Go to the section 'Compute Engine' > 'Instance groups' :

We need to create something like that:

Organise VM instances in a group to manage them together. [Instance groups](#) 

Name 



kubernetes-engine

Description (Optional)

Autoscaling group for kuberentes engine cluster.


Location
To ensure higher availability, select a multiple zone location for an instance group.
[Learn more](#)

☒ Single zone
☐ Multiple zones


Region  **Zone** 

us-west1 (Oregon) us-west1-b

[Specify port name mapping](#) (Optional)

Instance template 

kubernetes-template

Auto-scaling 

On

Auto-scaling policy

Auto-scaling policy
Use metrics to determine when auto-scaling should resize the group.
[Auto-scaling groups of instances](#)

CPU usage

Target CPU usage ?
60 %

Minimum number of instances ?
3

Maximum number of instances ?
5

Cool-down period ?
30 seconds

Autohealing ?
To use auto-healing, configure firewall rules. This will allow health checks to connect to VM instances in a group.
[How to configure firewall rules](#)

Health check
No health check
Compute Engine will recreate VM instances only when they're not running.

[Advanced creation options](#)

NOTE: the auto-scaling policy depends of your infrastructure and your necessity.

We create the group of instances and wait for them to be created. Then we will connect using the google sdk as we did in the previous sections. We will also check that the pre-installation script that we added has been executed correctly.

```
anonymous@bastion ~]$ ssh master
Last login: Thu May 30 09:24:43 2019 from bastion.c.sturdy-apricot-241218.internal
[anonymous@kubernetes-engine-7cgh ~]$
```

Instance groups [CREATE INSTANCE GROUP](#) [REFRESH](#) [DELETE](#)

Filter resources								Columns
<input type="checkbox"/> Name ^	Zone	Instances	Template	Creation time	Recommendation	Auto-scaling	In use by	
<input checked="" type="checkbox"/> kubernetes-engine	us-west1-b	3	kubernetes-template	15 May 2019, 11:32:36		Target CPU usage 60%		

VM instances								CREATE INSTANCE	IMPORT VM	REFRESH	START	STOP	RESET	DELETE
Filter VM instances								Columns						
<input type="checkbox"/> Name ^	Zone	Recommendation	In use by	Internal IP	External IP	Connect								
<input checked="" type="checkbox"/> kubernetes-engine-kd3n	us-west1-b		kubernetes-engine	10.138.0.48 (nic0)	35.197.0.234	SSH								
<input checked="" type="checkbox"/> kubernetes-engine-s9xv	us-west1-b		kubernetes-engine	10.138.0.47 (nic0)	34.83.97.141	SSH								
<input checked="" type="checkbox"/> kubernetes-engine-sn7d	us-west1-b		kubernetes-engine	10.138.0.49 (nic0)	35.247.34.46	SSH								

```
[anonymous@kubernetes-engine-cc7h ~]$ cat /startup-5y5qFH/
cat: /startup-5y5qFH/: Permiso denegado
[anonymous@kubernetes-engine-cc7h ~]$ sudo su
[root@kubernetes-engine-cc7h anonymous]# cat /startup-5y5qFH/
cat: /startup-5y5qFH/: Es un directorio
[root@kubernetes-engine-cc7h anonymous]# cat /startup-5y5qFH/tmpdLdHkf
#!/bin/bash
yum -y update && yum -y update && yum -y install make gcc kernel-devel kernel-headers net-tools vim
setenforce 0
sed -i --follow-symlinks 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/selinux
modprobe br_netfilter
echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables
swapoff -a
yum install -y yum-utils device-mapper-persistent-data lvm2
yum-config-manager \
--add-repo \
https://cloudlinux.com/updates/updates.repo
```

We check that the script was created and executed successfully and the demons are working.

```
[root@kubernetes-engine-lrz6 anonymous]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since mié 2019-05-15 11:14:54 UTC; 4min 56s ago
     Docs: https://docs.docker.com
   Main PID: 26981 (dockerd)
    Tasks: 9
   Memory: 29.3M
   CGroup: /system.slice/docker.service

[root@kubernetes-engine-lrz6 anonymous]# systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
   Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; vendor preset: disabled)
   Drop-In: /usr/lib/systemd/system/kubelet.service.d
           └─10-kubeadm.conf
   Active: activating (auto-restart) (Result: exit-code) since mié 2019-05-15 11:19:56 UTC; 9s
     Docs: https://kubernetes.io/docs/
```

Once we have the template and autoscaler of the kubernetes nodes, we will proceed to assemble the cluster with kubeadm.

To setup the cluster, follow the previous section '**CREATING THE CLUSTER WITH KUBEADM**'.

NOTE: you can use any instance created as master. From here, I recommend signing the name/hostname of the instance to later interact with the master since it is the most important.

Obviously if the master instance falls, the autoscaler will create a new one and the master will change the hostname and get a new ip. But, it is difficult for the master to stop working. However, we have the autoscaler to control this.

THINGS TO KEEP IN MIND:

If we reduce or stop the instances of our autoscaler, the configuration of the kubernetes cluster will be lost. In case of creating new instances of those that we already have, we would have to configure by hand the 'token' (kubeadm join) for the new worker of Kubernetes. The rest of pre-installed services are already configured with the template kubernetes-template.

It is possible to automate the incorporation of kubernetes nodes automatically, but this will not be done as it is a higher level step. For example, the GKE kubernetes service that Google incorporates, automates it, although I prefer to do everything manually since it is better to learn.

What happens if I lose the token when creating the cluster with kubeadm?

I can create new token in the master with the following command:

```
$ kubeadm token create --print-join-command
```

IMPORTANT: It is very necessary because if we need to add a new node (it will be added by our autoscaler) we will have to add the token manually since this action is not automated as it was explained in the previous paragraph.

More info: <https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm-join/>

CREATION OF OUR PERSISTENT FILE SYSTEM WITH GLUSTERFS

What is Gluster ?

Gluster is a scalable, distributed file system that aggregates disk storage resources from multiple servers into a single global namespace.

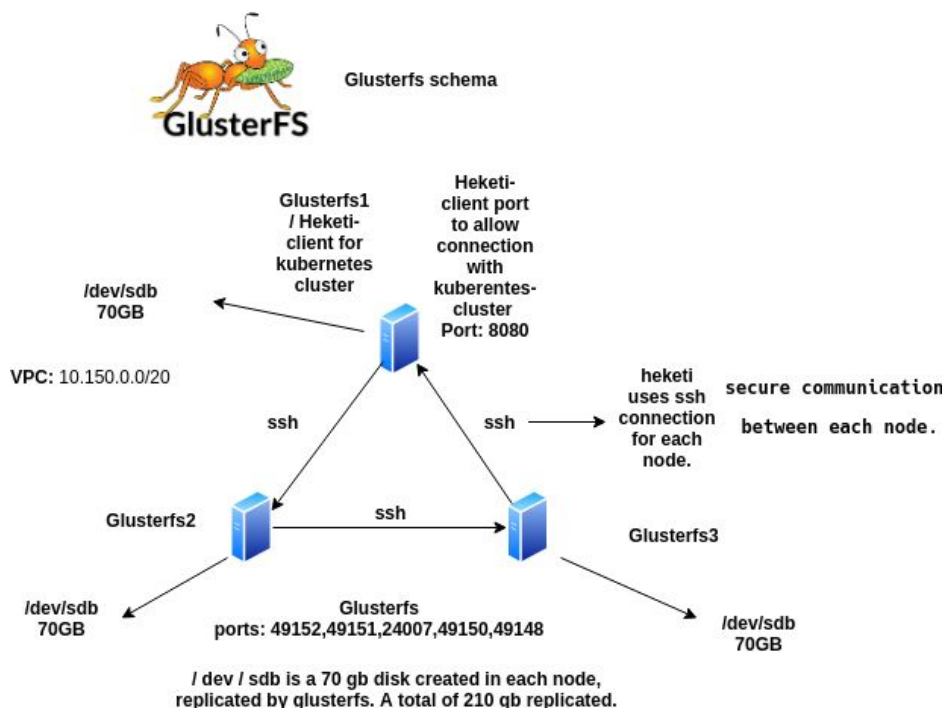
Why did we choose glusterfs?

It is an easy to assemble and efficient system. To mount a cluster of glusterfs is quite simple and the replication of the files is effective. It is an example of HA (High Availability) cluster.

What client will we use to communicate glusterfs with the kubernetes cluster?

Heketi provides a RESTful management interface which can be used to manage the life cycle of GlusterFS volumes. With Heketi, cloud services like OpenStack Manila, Kubernetes, and OpenShift can dynamically provision GlusterFS volumes with any of the supported durability types. Heketi will automatically determine the location for bricks across the cluster, making sure to place bricks and its replicas across different failure domains. Heketi also supports any number of GlusterFS clusters, allowing cloud services to provide network file storage without being limited to a single GlusterFS cluster.

Let's see our scheme:



Server	OS	Disk1 - filesystem 'p'	Disk2 - replicated	RAM - CORES
Glusterfs1	Centos 7	10GB -HDD	70GB -HDD	3.75GB - 1 CORE
Glusterfs2	Centos 7	10GB -HDD	70GB -HDD	3.75GB - 1 CORE
Glusterfs3	Centos 7	10GB -HDD	70GB -HDD	3.75GB - 1 CORE

Creation of the machines in Google Cloud Platform:

IMPORTANT: the process of creating the instances is not going to do so since in previous sections it is already documented. You just have to take into account when adding the 70GB disk in each instance in the following section:

Administración Seguridad **Discos** Redes Único cliente

Disco de arranque
Regla de eliminación
☒ Eliminar el disco de arranque cuando se elimine la instancia



Encriptado
Los datos se encriptan automáticamente. Selecciona una solución para administrar claves de encriptado.

☒ Clave administrada por Google
No se requiere configuración

☐ Clave administrada por el cliente
Administración a través de Google Cloud Key Management Service

☐ Clave proporcionada por el cliente
Administración fuera de Google Cloud

Discos adicionales ? (Opcional)

Disco nuevo (disk-1, en blanco, 500 GB)  

Nombre (Opcional) ?

Descripción (Opcional)

Tipo ?
Disco persistente estándar ▼

Tipo de origen ?
☒ Disco en blanco ☐ Imagen

Modo
☒ Lectura/Escritura
☐ Solo lectura

Regla de eliminación
Al eliminar la instancia
☒ Conservar disco
☐ Eliminar disco

Tamaño (GB) ?

Also, we have to keep in mind that we must add the following automation script at startup:
https://github.com/nanih98/iamcloud/blob/master/Scripts/nfs_gcp_script.sh

Finally, add the ssh-bastion firewall rule to enable ssh connection from bastion server:

Administración **Seguridad** Discos Redes Único propietario

VM blindada ⓘ
Para usar los roles de VM blindada, debes seleccionar una imagen blindada.
Para disfrutar de la configuración más segura, activa todos los ajustes.

☐ Activa el arranque seguro ⓘ
☐ Activa el vTPM ⓘ
☐ Activa la supervisión de integridad ⓘ

Claves SSH
Estas claves permiten acceder únicamente a esta instancia, a diferencia de las [claves SSH del proyecto](#). [Más información](#)

☐ **Bloquear claves SSH del proyecto**
Si marcas esta opción, las claves SSH del proyecto no pueden acceder a esta instancia. [Más información](#)

anonymous

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCAQC7nNkhH0aNI/K6pB7Ub844iIwmTTB0aNsh/H0PMnULX7U13JQqqy314EhWy0RmsQCZEhvt8oaEtsxt9Zc1c2TWXHdn86wUu/82yvoRM0hxhCOiWKqpodhb1x091Q+cCj9bwGJUfo00uksUKx4NkOj0o0FgDWyj35nkraYN1o/P1rryJxMBQEY5no+GDv1MDEfeCuapveXjI/19u31zw1rR2mrQPBhk1pL/BiEioF1vciYwQ0/TKQaaJFXbmTpVN1A
```

[+ Añadir elemento](#)

HEKETI INSTALLATION

As we explained earlier, he chose heketi to manage our volumes of glusterfs, because from kubernetes we can automate the deployment of volumes without having to interact with the nodes of glusterfs.

Step 1: Configure ssh 'root' acces for glusterfs connection.

Heketi communicates with the other nodes through ssh. In this case, we will enable the root user (not secure) to do the tests.

```
root@server1:# vi /etc/ssh/sshd_config
PermitRootLogin yes
```

```
root@server1:# vi /etc/selinux/config
SELINUX=disabled
```

```
root@server1:# setenforce 0
```

Do the same on each nodes.

Create a new firewall rule to enable connection to port 22 from the vpc of the glusterfs cluster:



Apply this firewall rule label to all the glusterfs nodes.

1.1: Connect the nodes with 'gluster peer probe 'gluster_name' '

```
root@server1:~# gluster peer probe server2
peer probe: success.
root@server1:~# gluster peer status
Number of Peers: 1
Hostname: server2
State: Peer in Cluster (Connected)
```

Do the same on each nodes.

Step 2: Heketi installation and configuration

```
root@server1:~# wget
https://github.com/heketi/heketi/releases/download/v8.0.0/heketi-v8.0.0.linux.amd64.tar.gz
root@server1:~# tar xzvf heketi-v8.0.0.linux.amd64.tar.gz
root@server1:~# cd heketi
root@server1:~# cp heketi heketi-cli /usr/local/bin/
root@server1:~# heketi -v
```

```
root@server1:~# groupadd -r -g 515 heketi
root@server1:~# useradd -r -c "Heketi user" -d /var/lib/heketi -s /bin/false -m -u 515 -g heketi heketi
root@server1:~# mkdir -p /var/lib/heketi && chown -R heketi:heketi /var/lib/heketi
root@server1:~# mkdir -p /var/log/heketi && chown -R heketi:heketi /var/log/heketi
root@server1:~# mkdir -p /etc/heketi
```

Heketi has several provisioners but here I will be using the ssh. We need to set up password-less ssh login between the Gluster nodes so heketi can access them. Generate RSA key pair.

```
root@server1:~# ssh-keygen -f /etc/heketi/heketi_key -t rsa -N ""
root@server1:~# chown heketi:heketi /etc/heketi/heketi_key*
```

Apply the public key in each nodes:



Create the Heketi config file in /etc/heketi/heketi.json:

```
root@server1:~# vim /etc/heketi/heketi.json
```

Config file: https://github.com/nanih98/iamcloud/blob/master/Heketi_config/heketi.json

Create the following Heketi service file /etc/systemd/system/heketi.service:

```
root@server1:~# vim /etc/systemd/system/heketi.service
```

Config file: https://github.com/nanih98/iamcloud/blob/master/Heketi_config/heketi.service

```
root@server1:~# systemctl daemon-reload
root@server1:~# systemctl start heketi.service
root@server1:~# journalctl -xe -u heketi
root@server1:~# systemctl enable heketi
```

```
[root@glusterfs1 ~]# systemctl status heketi
● heketi.service - Heketi Server
   Loaded: loaded (/etc/systemd/system/heketi.service; disabled; vendor preset: disabled)
   Active: active (running) since lun 2019-05-27 14:34:23 UTC; 6 days ago
     Main PID: 28888 (heketi)
        Tasks: 7
       Memory: 3.2M
      CGroup: /system.slice/heketi.service
              └─28888 /usr/local/bin/heketi --config=/etc/heketi/heketi.json

jun 02 16:14:24 glusterfs1 heketi[28888]: Memory: 238.1M
```

Create topology/etc/heketi/topology.json config file:

```
root@server1:~# vim /etc/heketi/topology.json
```

Config file: https://github.com/nanih98/iamcloud/blob/master/Heketi_config/topology.json

IMPORTANT: the configuration of the topology.json file must be adapted to your infrastructure. Where /dev/sdb is a 70GB raw block device attached to each gluster node.

```
[root@server1 ~]# export HEKETI_CLI_SERVER=http://server1:8080
[root@server1 ~]# export HEKETI_CLI_USER=admin
[root@server1 ~]# export HEKETI_CLI_KEY=PASSWORD
```

If everything went well, you should show us the following:

```
root@ip-10-99-3-216:/opt/heketi# heketi-cli topology load --json=/opt/heketi/topology.json
Found node glustera.tftest.encompasshost.internal on cluster 37cc609c4ff862bfa69017747ea4aba4
Adding device /dev/xvdf ... OK
Found node glusterb.tftest.encompasshost.internal on cluster 37cc609c4ff862bfa69017747ea4aba4
Adding device /dev/xvdf ... OK
Found node glusterb.tftest.encompasshost.internal on cluster 37cc609c4ff862bfa69017747ea4aba4
Adding device /dev/xvdf ... OK
```

```
[root@server1 ~]# heketi-cli cluster list
Clusters:
Id:d1694da0ea9710c9ab44829db617094d [file][block]
```

```
[root@server1 ~]# heketi-cli node list
Id:2bcc7da8d6d556062cd0f72901f2ee5e Cluster:d1694da0ea9710c9ab44829db617094d
Id:95ec22225d398a9e3fb2fd304e2ab370 Cluster:d1694da0ea9710c9ab44829db617094d
Id:ff3aeb28dcb2a6c61be7672b40bbea62 Cluster:d1694da0ea9710c9ab44829db617094d
```

Create new firewall rule to enable port 8080 (heketi-client) to the kubernetes subnet:

heketi-client-8080

Registros 

Desactivado

[view](#)

Red

default

Prioridad

1000

Dirección

Entrada

Acción tras coincidencia

Permitir

Destinos

Etiquetas de destino	
heketi-client-8080	

Filtros de origen

Intervalos de IP	
10.150.0.0/20	
10.138.0.0/20	

Protocolos y puertos

tcp:8080

udp:8080

Aplicación

Apply only in glusterfs1 since it is the machine that has installed heketi-cli.

CONFIGURATION OF KUBERNETES TO ACCESS HEKETI-CLI

Download the following files, modify them and adapt them to your configuration and finally apply them to the kubernetes cluster (do it in the master).

1. `secret-heketi.yaml`
2. `storageclass-heketi.yaml`

Config files: https://github.com/nanih98/iamcloud/tree/master/Kubernetes_files

```
$ kubectl create -f secret-heketi.yaml  
$ kubectl create -f storageclass-heketi.yaml
```

Now, we already have the connection established between the kubernetes cluster and the glusterfs cluster. The next step will be to create our deployments to launch our applications. Obviously these deployments will have a configuration that they will keep in persistent volumes of glusterfs.

DEPLOYING WORDPRESS

IMPORTANT: As an application we have chosen wordpress to launch as frontend. Whatever the application, the most important step is the creation of our pvc (Persistent Volume Claim), since this pvc is going to reserve a space of 'X' gb to our cluster of glusterfs. The request will be made to our client heketi.

Download the following files from the repo:

1. `secret-mysql.yaml`
2. `mysql-deployment-heketi.yaml`
3. `wordpress-deployment-heketi.yaml`

Config files: https://github.com/nanih98/iamcloud/tree/master/Kubernetes_files

Create them:

```
$ kubectl create -f secret-mysql.yaml -f mysql-deployment-heketi.yaml -f  
wordpress-deployment-heketi.yaml
```

We verify that the pvc have been created and our cluster of glusterfs have been correctly mapped:

```
[anonymous@kubernetes-engine-7cgh Prueba]$ kubectl get pvc
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
mysql-pv-claim2	Bound	pvc-38b3a741-8261-11e9-8028-42010a8a0014	5Gi	RWX	gluster-heketi-external	3d18h
wppv-claim2	Bound	pvc-38b61cc3-8261-11e9-8028-42010a8a0014	5Gi	RWX	gluster-heketi-external	3d18h

```
[anonymous@kubernetes-engine-7cgh Prueba]$
```

Executing the gluster volume list command in the gluster nodes should show us the two volumes that we created, 1 for mysql and the other for wordpress. Effectively they have been created and mapped correctly:

```
[root@glusterfs1 anonymous]# gluster volume list
vol_7bc3d1e031e3c3ed9b7cdfda448ca831
vol_cfd0fc189de750895bd3b31004576557
[root@glusterfs1 anonymous]#
```

```
[root@glusterfs1 anonymous]# ls /var/lib/heketi/mounts/vg_219caee39af153d10739ddde937d28c3/b
rick_
brick_272f3fa196a0b28f8e237bdd115fcd4c/ brick_c47a850dcca9b9186e16cd95058a6f6b/
[root@glusterfs1 anonymous]# ls /var/lib/heketi/mounts/vg_219caee39af153d10739ddde937d28c3/b
rick_c47a850dcca9b9186e16cd95058a6f6b/brick/
auto.cnf          ibdata1          ib_logfile1      performance_schema/
.glusterfs/       ib_logfile0      mysql/           test/
```

Check:

```
root@anonymous 09:36:54 /home/anonymous/Minikube/Wordpress → kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
jenkins	LoadBalancer	10.107.126.45	<pending>	8080:30717/TCP	4d
jenkins-agent	ClusterIP	10.101.216.93	<none>	50000/TCP	4d
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	5d
noodle-mariadb	ClusterIP	10.107.255.88	<none>	3306/TCP	4d
noodle-moodle	LoadBalancer	10.109.100.70	<pending>	80:30997/TCP,443:32442/TCP	4d
vordpress	LoadBalancer	10.96.152.249	<pending>	80:32316/TCP	1m
vordpress-mariadb	ClusterIP	10.103.74.128	<none>	3306/TCP	5d
vordpress-mysql	ClusterIP	None	<none>	3306/TCP	1m
vordpress-wordpress	LoadBalancer	10.105.187.55	<pending>	80:30745/TCP,443:31691/TCP	5d

```
root@anonymous 09:37:53 /home/anonymous/Minikube/Wordpress →
```

Once we have the released deployment of wordpress / mysql, exposed as a ClusterIP service, we will create the certificates with cert-manager using helm and later we will finish installing our loadbalancer.

INSTALL CERT-MANAGER WITH HELM

Cert-manager is a native Kubernetes certificate management controller. It can help with issuing certificates from a variety of sources, such as Let's Encrypt, HashiCorp Vault, Venafi, a simple signing keypair, or self signed.

Installation:

```
$ kubectl create namespace cert-manager
$ kubectl label namespace cert-manager certmanager.k8s.io/disable-validation=true
$ kubectl apply -f
https://github.com/jetstack/cert-manager/releases/download/v0.8.0/cert-manager.yaml
$ kubectl create clusterrolebinding cluster-admin-binding \
  --clusterrole=cluster-admin \
  --user=$(gcloud config get-value core/account)
```

Installing with helm:

```
$ helm repo add jetstack https://charts.jetstack.io
$ kubectl apply -f
https://raw.githubusercontent.com/jetstack/cert-manager/release-0.8/deploy/manifests/00-crd
s.yaml
$ helm install --name cert-manager --namespace cert-manager jetstack/cert-manager
```

Now let's create a certificate using let's encrypt with cert-manager for our domain wordpress-iamcloud.ddns.net.

We will use two objects, issuer and certificate.

With the object issuer we will specify the url where our service will look for the certificate, in this case it is a url of let's encrypt.

With the certificate object, we will issue the certificate.

Let's see:

<https://github.com/nanih98/iamcloud/tree/master/Cert-manager>

Download the files, adapt them to your configuration and create them:

```
$ kubectl create -f issuer.yaml -f certificate.yaml
```



```
[anonymous@kubernetes-engine-7cgh front]$ kubectl get secrets -n front
```

NAME	TYPE	DATA	AGE
certificate-nginx	kubernetes.io/tls	3	5d6h
default-token-s6j7s	kubernetes.io/service-account-token	3	6d3h
letsencrypt	Opaque	1	5d6h

```
[anonymous@kubernetes-engine-7cgh front]$
```

Once we have the certificate, we must create our ingress to be able to redirect the incoming traffic by our http/s loadbalancer to our pods.

Download the ingress and create it from your configurations:

<https://github.com/nanih98/iamcloud/blob/master/Ingress/ingress-tls.yaml>

\$ kubectl create -f ingress-tls.yaml

```
[anonymous@kubernetes-engine-7cgh front]$ kubectl get ing
```

NAME	HOSTS	ADDRESS	PORTS	AGE
ingress-resource	nginx.example.com		80	5d22h

```
[anonymous@kubernetes-engine-7cgh front]$
```

CREATE INGRESS CONTROLLER WITH TRAEFFIK

Run the following commands to install traefik ingress controller:

\$ kubectl apply -f

<https://raw.githubusercontent.com/containous/traefik/v1.7/examples/k8s/traefik-deployment.yaml>

\$ kubectl apply -f

<https://raw.githubusercontent.com/containous/traefik/v1.7/examples/k8s/traefik-rbac.yaml>

Now we must know the port which our ingress will expose to create later our loadbalancer http/s.

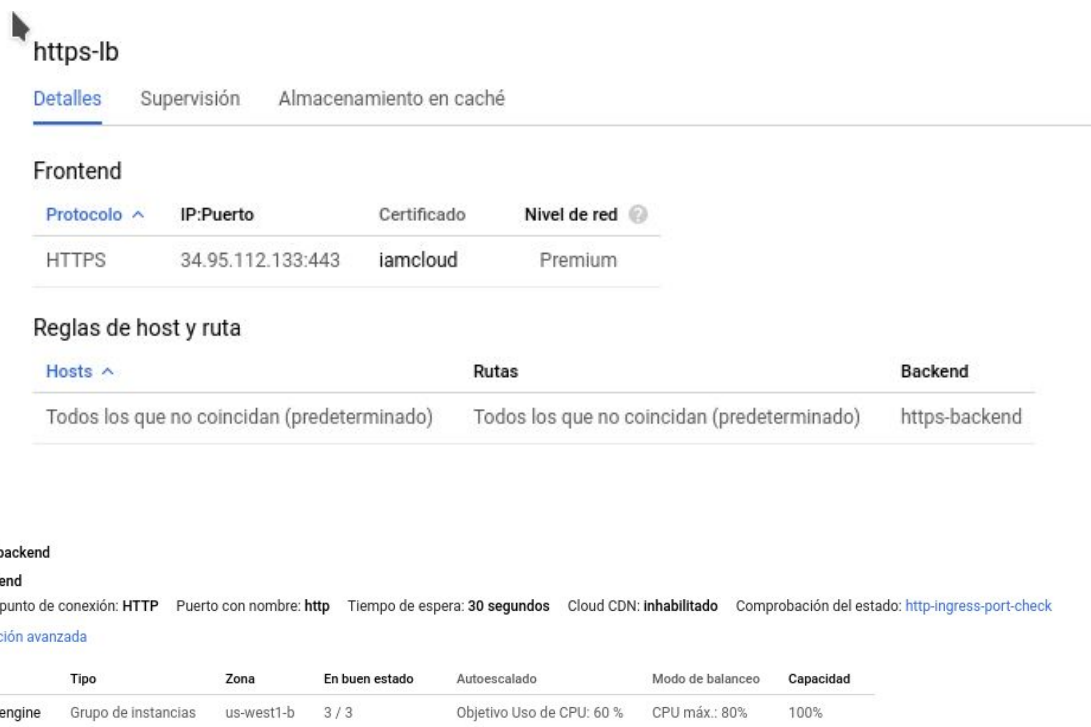
Run:

```
[anonymous@kubernetes-engine-7cgh front]$ kubectl get svc -n kube-system
NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)                                AGE
kube-dns                            ClusterIP           10.96.0.10      <none>           53/UDP,53/TCP,9153/TCP                7d3h
kubernetes-dashboard                NodePort            10.107.21.217   <none>           443:30620/TCP                        5d2h
tiller-deploy                       ClusterIP           10.108.135.106  <none>           44134/TCP                             6d7h
traefik-ingress-service             NodePort            10.104.64.151   <none>           80:30192/TCP,8080:32018/TCP          5d4h
[anonymous@kubernetes-engine-7cgh front]$
```

The important port we will need is the **30192** (as we can see in the previous image), so that our loadbalancer redirects the traffic to that port and later our ingress object that we had previously created, redirects the pods.

CREATE THE HTTP/S LOADBALANCER

We will mount a loadbalancer that redirects the traffic to the port that has been exposed by the ingress controller controller. Therefore our backend will be the kubernetes nodes. Specifically port **30192** as specified in the previous section.



https-lb

[Detalles](#) [Supervisión](#) [Almacenamiento en caché](#)

Frontend

Protocolo	IP:Puerto	Certificado	Nivel de red
HTTPS	34.95.112.133:443	iamcloud	Premium

Reglas de host y ruta

Hosts	Rutas	Backend
Todos los que no coincidan (predeterminado)	Todos los que no coincidan (predeterminado)	https-backend

Backend

Servicios de backend

1. https-backend

Protocolo de punto de conexión: HTTP Puerto con nombre: http Tiempo de espera: 30 segundos Cloud CDN: Inhabilitado Comprobación del estado: [http-ingress-port-check](#)

[Configuración avanzada](#)

Nombre	Tipo	Zona	En buen estado	Autoescalado	Modo de balanceo	Capacidad
kubernetes-engine	Grupo de instancias	us-west1-b	3 / 3	Objetivo Uso de CPU: 60 %	CPU máx.: 80%	100%

MONITORING OF SYSTEMS

MONITORING THE MAIN SERVICES WITH MONIT (EXTRA)

In order to monitor the most important services of our servers, we will use a very useful software called **monit**. This is the first type of monitoring we should do since they are critical system services that must be controlled.

The main services that we will use are:

- **Cron:** necessary to start services when the system is restarted.
- **Docker:** the main service when there are kubernetes engine and other containers.
- **Networking:** essential for Internet connectivity or other access.
- **SSH:** for remote connection against machines (only admin preferably).

NOTE: It is essential to have this software (monit) on all servers. Then the configuration of one or the other will change. Kubernetes are included in docker service because the services are containerized with docker as I explained before (kubeadm). Therefore, it is not necessary to control the 'kubelet' service itself.

Installation:

```
$ yum install monit
```

```
$ systemctl enable monit && systemctl start monit
```

Monit configuration:

Open the configuration file on **/etc/monitrc** and add:

```
#####Services#####
##Docker service
check process docker with pidfile /run/docker.pid
start program "/usr/bin/systemctl start docker"
stop program "/usr/bin/systemctl stop docker"
#
##Ssh service
check process sshd with pidfile /run/sshd.pid
start program "/usr/bin/systemctl start sshd"
stop program "/usr/bin/systemctl stop sshd"
if failed host 127.0.0.1 port 22 protocol ssh then restart
if 5 restarts within 5 cycles then timeout
#
##Networking service
check process networking with pidfile /run/dhclient-eth0.pid
start program "/usr/bin/systemctl start NetworkManager"
stop program "/usr/bin/systemctl stop NetworkManager"
#
#
##Cron service
check process crond with pidfile /run/crond.pid
start program "/usr/bin/systemctl start crond"
stop program "/usr/bin/systemctl stop crond"
#
#####
```

Also we have to uncomment the port configuration and the admin/password:

```
##
set httpd port 2812 and
##   use address localhost # only accept connection from localhost
##   allow localhost      # allow localhost to connect to the server and
allow admin:ausias38      # require user 'admin' with password 'monit'
#   #with ssl {           # enable SSL/TLS and set path to server certificate
##       # pemfile: /etc/ssl/certs/monit.pem
#       #}
#
```

Restart the service and put the public ip of the server and the port 2812:

\$ systemctl restart monit
url= publicip:2812

NOTE: don't remember to create the firewall rule to allow the port of the service monit to the instances and apply the label.



monit

Registros ?

Desactivado

[view](#)

Detalles de regla de cortafuegos

Red

default

Prioridad

1000

Dirección

Entrada

Acción tras coincidencia

Permitir

Destinos

Etiquetas de destino

monit

Filtros de origen

Intervalos de IP

85.192.70.145/32

Protocolos y puertos

tcp:2812

Aplicación

Habilitado

Follow the steps and do the same with each other.

MONITORING ALL THE INFRASTRUCTURE WITH GRAFANA METRICS

What is Grafana?

Grafana is a tool made in free software, specifically with Apache 2.0 license, designed by Torkel Ödegaard (who is still in the front of its development and maintenance) and created in January 2014. in 2012 (to date) it continues offering services of development and consulting in this popular private platform, in parallel with the development of free software.

What does Grafana?

Grafana is a tool to visualize temporary series data. From a series of collected data we will obtain a graphic overview of the situation of a company or organization.

The sole objective of Grafana is to present the monitoring data in a more user-friendly and pleasant way. At this point we must make a clarification: you can collect data from Cloudwatch, Graphite, Elasticsearch, OpenTSDB, Prometheus, Hosted Metrics and InfluxDB natively.

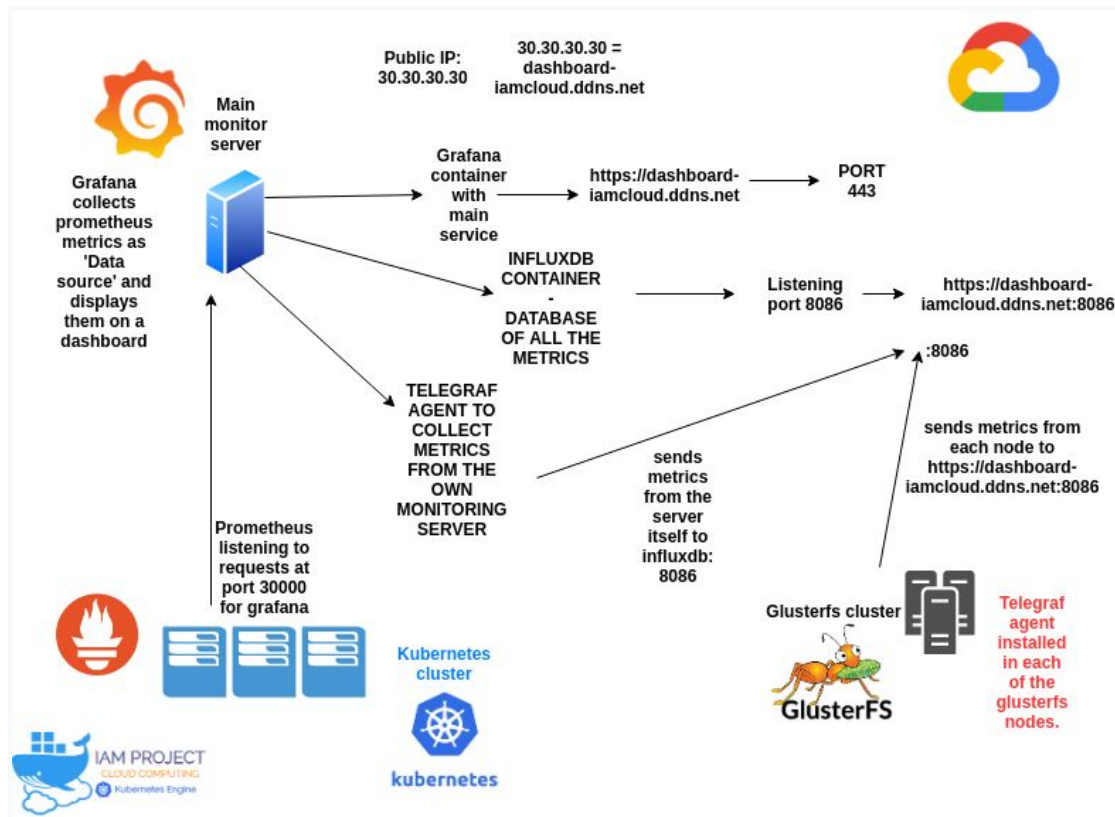
- **37 complements of data sources.**
- **28 complements for the panel.**
- **15 application complements.**
- **More than 600 control panels created for popular applications.**

More info: <https://grafana.com/>

In our infrastructure we will use the following technologies:

1. **Telegraf:** will be the agent that will be installed in each of the servers (Docker container), responsible for sending the state metrics to the influxdb database.
2. **Influxdb:** InfluxDB is an open-source time series database (TSDB) developed by InfluxData. It is written in Go and optimized for fast, high-availability storage and retrieval of time series data in fields such as operations monitoring, application metrics. Will be used as 'data endpoint', from our grafana server will collect the data that previously telegraf has sent. It is an example of Big Data, since thousands of data are processor per minute, depending on the number of servers that we have monitoring.
3. **Prometheus:** Prometheus is an open-source software application used for event monitoring and alerting. It records real-time metrics in a time series database built using a HTTP pull model, with flexible queries and real-time alerting. It is widely used to collect node metrics from a kubernetes cluster. We can consult a lot of metrics and consult graphics from the prometheus graphical interface itself. But, since we use grafana as the main monitoring server, we will also use prometheus as a 'data endpoint' to graphically show all the data collected by prometheus. Therefore, we will have two ways of consulting metrics from our kubernetes cluster.
4. **Grafana:** the main server as we have commented, mounted on a centralized monitoring server. All applications are mounted with Docker, except Prometheus which is installed by launching a Kubernetes deployment.
5. **Telegram:** we will use this 'secure' messaging application. We will create a bot and a group to be able to receive grafana alerts in a safe way. Grafana accepts several 'alerting channels', like emails, slack ... etc In our case we have chosen telegram since the mobile phone is something that we usually carry over. However, it would be better to set up another system of alerts (not graphics), something like Nagios or OP5 to be able to centralize alerts.

This is the final result of our scheme:



Now we proceed to configure our monitoring server:

Create a new instance with the follow specs:

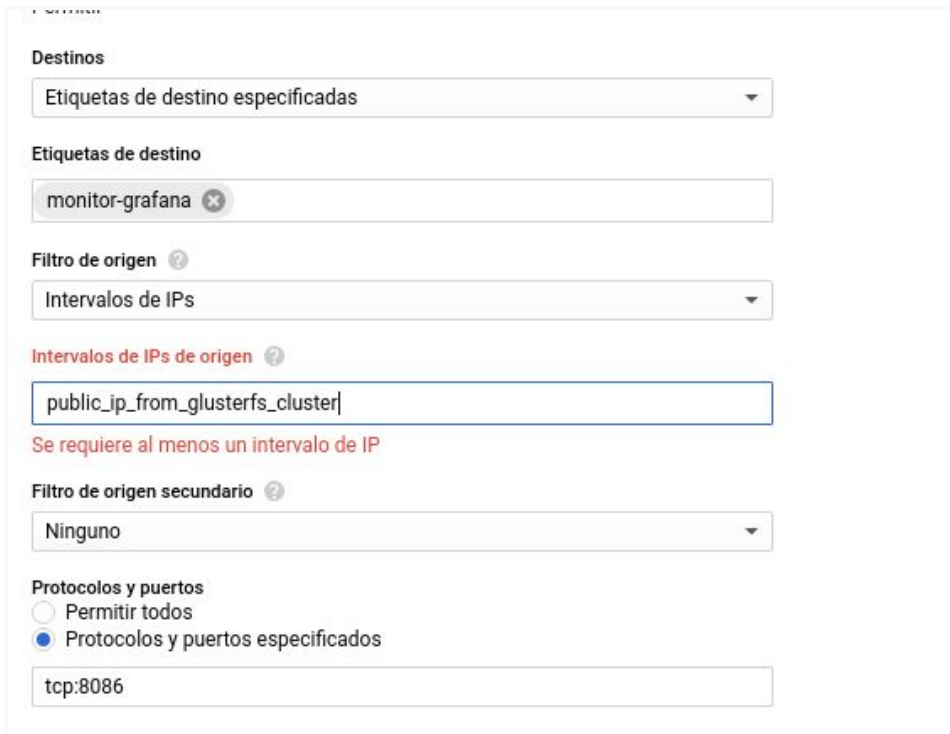
	Disk	RAM	CORE
Monitor	13GB SSD	6 GB	2 CORE

Then run the pre-script to install docker:

https://github.com/nanih98/iamcloud/blob/master/Monitoring_files/docker_installation.sh

Apply the follow labels:

1. **ssh-bastion: allow ssh connection from bastion instance**
2. **monitor-grafana: allow connection to influxdb database from public ip of glusterfs cluster. Necessary for monitor glusterfs metrics.**



Once we have installed the prerequisites, we launch our applications in container, for it, we will download the following files and launch the docker-compose:

1. env.grafana
2. env.influxdb
3. run.sh
4. telegraf.conf
5. docker-compose

Then run:

\$ docker-compose up -d to run the containers.

```
[root@monitor Prueba]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED            STATUS             PORTS
47adb78e92d2       grafana/grafana:latest  "/run.sh"               3 days ago        Up 2 days         0.0.0.0:443->3000/tcp
a308ccde99f3       telegraf:latest     "/entrypoint.sh tele..." 6 days ago        Up 2 days         8092/udp, 8125/udp, 8094/tcp
22d0f1d68c7a       influxdb:latest     "/entrypoint.sh infl..." 6 days ago        Up 2 days         0.0.0.0:8083->8083/tcp,
0.0.0.0:8086->8086/tcp, 0.0.0.0:8090->8090/tcp
[root@monitor Prueba]#
```

IMPORTANT STEP: install docker on each glusterfs node and run telegraf container to send metrics to monitor server 8086 port.

Once we have docker installed in each server of glusterfs (using the script that I mentioned in the previous section), we execute the container telegraf using the configuration telegraf.conf that we download from the repository changing the following:

```
# urls = ["udp://localhost:8089"] # UDP endpoint example
urls = ["http://influxdb:8086"] # required
## The target database for metrics (telegraf will create it if not exists).
database = "telegraf" # required
```

Change <http://influxdb:8086> and put the public ip of the monitor instance to send the metrics.

DEPLOYING PROMETHEUS ON KUBERNETES CLUSTER

To deploy prometheus create the following scripts from my repository:

https://github.com/nanih98/iamcloud/blob/master/Monitoring_files/prometheus-deployment.yaml


\$ kubectl create -f prometheus-deployment.yaml

GRAFANA CONFIGURATION WITH DASHBOARDS

Add the 'data source' influxdb database. It is the container that is running on the same monitoring server.

Then, import a dashboard:

The dashboard number model is 914.



Import

Import dashboard from file or Grafana.com

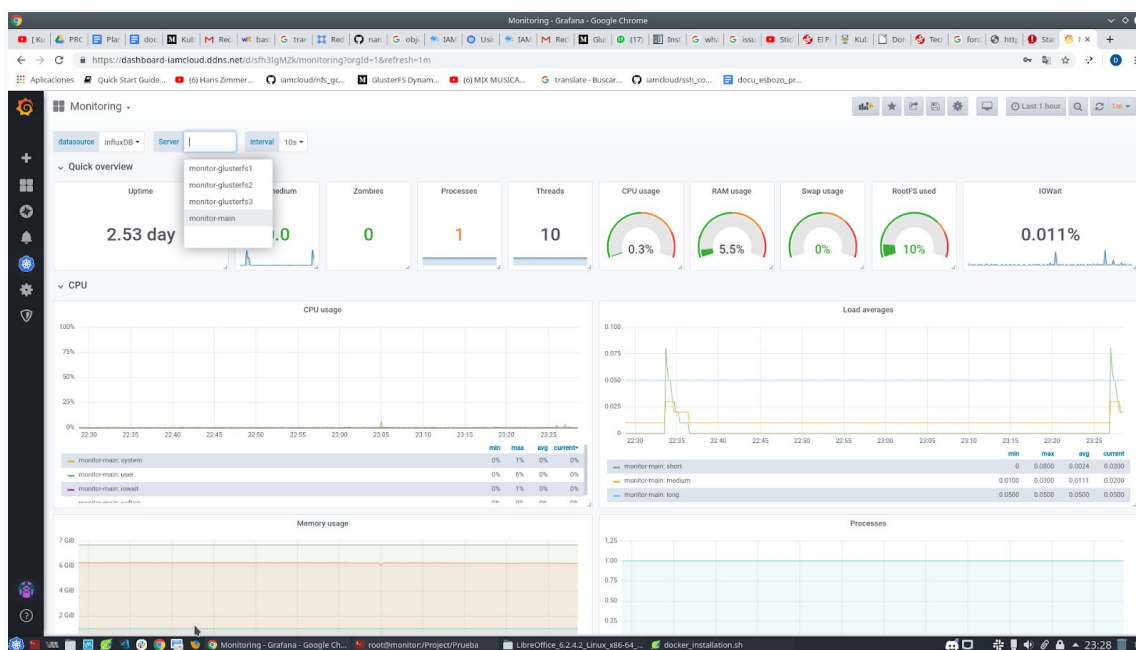
Importing Dashboard from [Grafana.com](#)

Published by	Lex Rivera
Updated on	2017-06-05 13:42:18

Options

Name	Telegraf system overview
Folder	General
Unique Identifier (uid)	auto-generated
InfluxDB telegraf	InfluxDB

Import
Cancel



Once we have monitored the cluster of glusterfs and the server itself, we must add as a data endpoint prometheus. For this we have a public ip of prometheus listening by port 30000.

Data Sources / Prometheus

Type: Prometheus

[Settings](#) [Dashboards](#)

Name Default ☐

HTTP

URL

Access [Help](#)

Whitelisted Cookies

Auth

Basic Auth ☐ With Credentials ☐

TLS Client Auth ☐ With CA Cert ☐

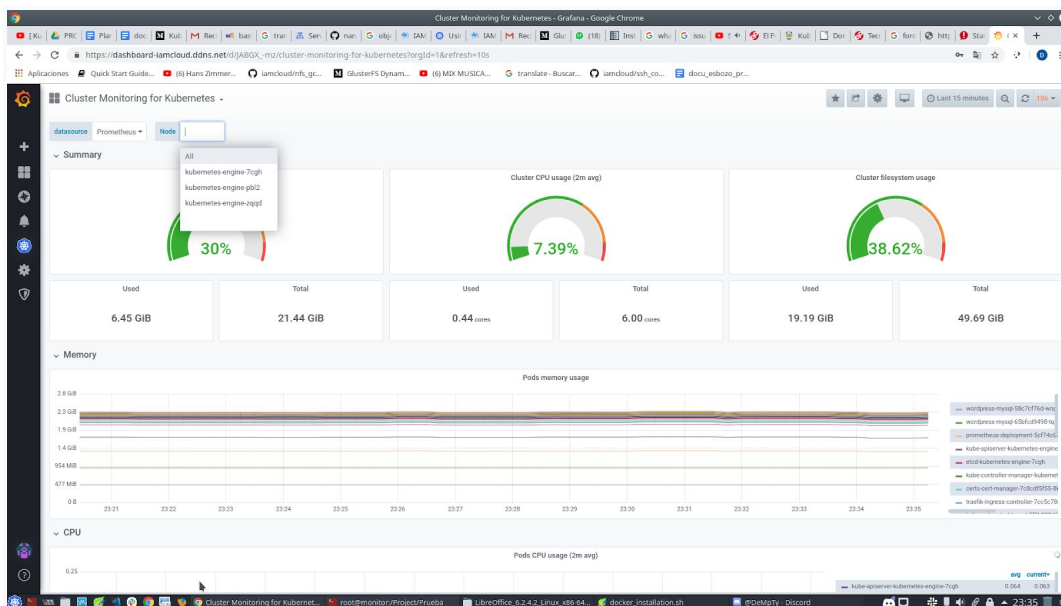
Skip TLS Verify ☐

Forward OAuth Identity ☐

Scrape interval

Query timeout

HTTP Method



CONCLUSIONS

It has been a very innovative project, which we have enjoyed a lot riding. I think it is a test facing the professional world, since nowadays it is betting on the cloud and using isolated applications with containers (docker + kubernetes). Also the monitoring software that we have used is one of the most pioneering of the market.

Finally, being honest, we have missed all the continuous CI / CD intricacy with Jenkins and Gitlab. Although he has not had time to present it, the iamcloud project will be incorporated as well. It would also have been interesting to build an elastic stack with a Kibana logstash.

WEBGRAPHY

Kubernetes main page (all the documentation):

<https://kubernetes.io/es/docs/>

NFS:

<https://www.jorgedelacruz.es/2017/12/26/kubernetes-volumenes-nfs/>

<https://www.lisenet.com/2016/setup-nfs-server-on-centos-7-and-configure-client-automount/>

<https://www.digitalocean.com/community/tutorials/how-to-set-up-an-nfs-mount-on-centos-6>

<http://web.mit.edu/rhel-doc/3/rhel-sag-es-3/s1-nfs-mount.html>

Cert-manager:

<https://docs.cert-manager.io/en/latest/getting-started/install/kubernetes.html>

Wordpress:

<https://medium.com/@containerum/how-to-deploy-wordpress-and-mysql-on-kubernetes-bda9a3fdd2d5>

<https://kubernetes.io/docs/tutorials/stateful-application/mysql-wordpress-persistent-volume/>

Namespaces:

<https://cloud.google.com/blog/products/gcp/kubernetes-best-practices-organizing-with-namespaces>

Kubernetes NodePort vs LoadBalancer vs Ingress? When should I use what? :

<https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0>

NFS cluster configurations:

<https://www.rediris.es/jt/jt2018/programa/jt/ponencias/?id=jt2018-jt--a25b1c3.pdf>

<https://snjksh.wordpress.com/2017/06/13/configure-nfs-high-availability-cluster-on-centos-7/>

<https://www.theurbanpenguin.com/create-3-node-drbd-9-cluster-using-drbd-manage/>

https://www.linbit.com/downloads/tech-guides/HA_NFS_Cluster_using_Pacemaker_and_DRBD_on_RHEL7.pdf

<https://www.linuxtechi.com/configure-nfs-server-clustering-pacemaker-centos-7-rhel-7/>

https://www.suse.com/documentation/sle-ha-12/pdfdoc/book_sleha_techguides/book_sleha_techguides.pdf

Glusterfs configuration:

<https://www.linuxtechi.com/setup-glusterfs-storage-on-centos-7-rhel-7/>

<https://docs.gluster.org/en/v3/Quick-Start-Guide/Quickstart/>

<https://wiki.centos.org/HowTos/GlusterFSonCentOS>

NFS GANESHA:

https://gluster-documentations.readthedocs.io/en/latest/Features/glusterfs_nfs-ganesha_integration/

<https://www.insentra.com.au/gluster-with-nfs-ganesha/>

https://access.redhat.com/documentation/en-US/Red_Hat_Storage/2.1/html/Administration_Guide/gluster-nfs_and_kernel-nfs_services.html

Traefik ingress kubernetes:

<https://docs.traefik.io/user-guide/kubernetes/>

Heketi installation:

<https://medium.com/searce/glusterfs-dynamic-provisioning-using-heketi-as-external-storage-with-gke-bd9af17434e5>

https://docs.okd.io/1.5/install_config/storage_examples/containerized_heketi_with_dedicated_gluster.html

<https://icicimov.github.io/blog/virtualization/Kubernetes-shared-storage-with-external-GlusterFS-backend/>



IAM CLOUD

Daniel Cascales